VALTER MONTEIRO OLIVEIRA JUNIOR

ESCALONAMENTO E OTIMIZAÇÃO SOB RESTRIÇÕES DE BARRAMENTOS

FLORIANÓPOLIS – SC

2004

UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

VALTER MONTEIRO OLIVEIRA JUNIOR

ESCALONAMENTO E OTIMIZAÇÃO SOB RESTRIÇÕES DE BARRAMENTOS

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Orientador:

LUIZ CLÁUDIO VILLAR DOS SANTOS

Florianópolis, Fevereiro de 2004.

ESCALONAMENTO E OTIMIZAÇÃO SOB RESTRIÇÕES DE BARRAMENTOS

Valter Monteiro Oliveira Junior

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação – área de conhecimento: Sistemas de Computação, e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Raul Sidnei Wazlawick (coordenador)

Banca Examinadora

_____

Luiz Cláudio Villar dos Santos (orientador)

_____

César Albenes Zeferino

_____

Olinto José Varela Furtado

_____

Luís Fernando Friedrich

“Pedras grandes, as vemos. Nas pequenas reside o risco do tropeço.”
adaptação de provérbio japonês

Reconhecimentos

Antes:
Mara Mariotti, Vera Schumacher, Sérgio Silveira, colegas da IONICS, que
me reaproximaram da universidade, e me ajudaram a vislumbrar
um novo caminho cheio de oportunidades
para o meu crescimento pessoal
e mudança de rumo profissional;

Durante:
Colegas do LAPS – Dione, Felipe, Henrique (*que força fish!*), e toda
a turma do mestrado – Leo, Lu, Déia, Fabricia, Taís e Daniel, sempre por perto,
sempre companheiros, sempre em festa e sempre otimistas,
Paty (*pelo sabadão de trabalho*) e Bruna (*por tolerá-lo*),
e à Verinha, secretária do PPGCC, muito mais que profissional, "é*x demaix!"*;

Depois:
A todos os novos amigos – nem ouso citar nomes pra
não correr risco de criar desavenças(*!*), que me ajudem
a fazer de cada "depois" um novo recomeço!

Agradecimentos Especiais

Ao Professor e orientador Luiz Cláudio,
por aceitar minha inclusão neste grupo de pesquisa,
provendo todos os subsídios e principalmente, as orientações.
Aos membros desta banca, Professores Olinto, Friedrich e César, pela disposição
em contribuir e participar!
Ao Evandro Silva, que sempre acreditou (algumas vezes mais que eu!)
no meu potencial, e me ajudou a encarar com simplicidade
os percalços - que a mim pareciam intransponíveis - durante o mestrado
(*Uau, que forte hein meu caro! Parece coisa das "humanas"...*),
e à Jaque, distante, porém sempre presente!

ESCALONAMENTO E OTIMIZAÇÃO SOB RESTRIÇÕES DE BARRAMENTOS

Valter Monteiro Oliveira Junior

Fevereiro / 2004

Orientador: Luiz Cláudio Villar dos Santos.

Área de Conhecimento: Ciências da Computação.

Palavras-Chave: Síntese de Alto Nível, Escalonamento, Restrições de Conexão.

Número de Páginas: 56

Resumo

Esta dissertação aborda o problema de escalonamento sob restrições de recursos em Síntese de Alto Nível. Tradicionalmente, os algoritmos de escalonamento associam operações a instantes de tempo, levando em conta um número pré-fixado de unidades funcionais (somadores, ALUs, multiplicadores). Entretanto, para viabilizar a execução de uma operação em uma unidade funcional, os operandos precisam ser preliminarmente lidos de registradores e transportados, através de barramentos, até as entradas da unidade funcional. Além disso, o resultado da operação precisa também ser transportado, através de um barramento, até o registrador destino. Conseqüentemente, o escalonamento de muitas operações em paralelo pode levar à alocação de um número proibitivo de barramentos. Isto torna desejável que um algoritmo de escalonamento seja capaz de manipular também restrições impostas por um número pré-fixado de barramentos. Este trabalho estende um algoritmo de escalonamento clássico, usando a noção de transferência entre registradores (RT) ao invés da simples noção de operação. Assim, o escalonador estendido torna-se capaz de manipular, além das restrições de precedência, restrições de recursos impostas por um número limitado de recursos, sejam eles unidades funcionais e/ou barramentos. Resultados experimentais mostram o impacto do número limitado de barramentos sobre a latência.

SCHEDULING AND OPTIMIZATION UNDER BUS CONSTRAINTS

Valter Monteiro Oliveira Junior

February / 2004

Advisor: Luiz Cláudio Villar dos Santos.
Area of Concentration: Computing Systems.
Keywords: High-level Synthesis, Scheduling, Bus constraints.
Number of Pages: 56

Abstract

This dissertation addresses the resource-constrained scheduling problem in High-Level Synthesis. Traditionally, scheduling algorithms map operations to time steps taking into account a predefined number of functional units (adders, ALUs, multipliers) in the datapath. However, in order to enable the execution of an operation, the operands must be read from the register file and be transferred through the buss to the proper functional unit. Besides, the result of the operation must be transferred to the destination register. Therefore, the parallel execution of several operations may lead to the allocation of a prohibitive number of buses. This work extends a classical scheduling algorithm, utilizing the notion of register transfer (RT), instead of the bare notion of operation. As a result, the extended scheduler is able to handle not only precedence and functional-unit constraints, but also the constraints imposed by a limited number of buses.

Experimental results show the impact of a limited number of buses on the latency.

Sumário

Lista de Figuras

Lista de Algoritmos

Lista de Tabelas

Lista de Gráficos

Lista de Acrônimos

ALU	Arithmetic and Logic Unit
ASIC	Application-Specific Integrated Circuits
CAD	Computer-Aided Design
DFG	Data Flow Graph
DPG	Data Path Graph
EDA	Eletronic Design Automation
HDL	Hardware Description Language
HLS	High-Level Synthesis
RF	Register File
RT	Register Transfer
SMG	State Machine Graph
SoC	System on Chip
UF	Unidade Funcional
VLSI	Very Large Scale Integration

1 Introdução

O crescimento da utilização de computadores e a também crescente credibilidade neles são resultado de pesquisas intensas nas mais diversas áreas. Cada vez mais, operações consideradas críticas estão sendo entregues a sistemas automatizados, tais como sistemas de controle e orientação de mísseis, controle de tráfego aéreo e rodoviário, até sistemas médicos de controle e monitoração de pacientes - auxiliando na sua longevidade -, onde falhas podem causar grandes perdas, inclusive financeiras ou de oportunidades.

Esses equipamentos estão sendo criados com a tecnologia de *Sistemas Embutidos* ("Embedded Systems"), ou seja, sistemas de controle e computação dedicados a uma aplicação [2]. A mais restritiva visão de um sistema embutido é um micro-controlador ou processador rodando um programa fixo. De uma forma mais geral, um sistema embutido é composto por três componentes básicos: processador, memória e, possivelmente, um circuito de aplicação específica, conforme ilustra a Figura 1.1, adaptada da figura original em [1]. Sensores e atuadores permitem que o sistema se comunique com o ambiente externo.

A memória armazena os dados e o programa a ser executado no processador. Memórias e processadores são fabricados para servir como componentes de uso geral, enquanto que o circuito de aplicação específica ou ASIC ("Application-Specific Integrated Circuit") é um circuito talhado para as características particulares de uma única aplicação.

Sistemas Embutidos pertencem à classe de sistemas reativos, porque eles são feitos para reagir ao ambiente executando funções em resposta a estímulos de entrada. Em alguns casos, suas funções devem ser executadas dentro de uma janela de tempo previamente definido. Por isso às vezes são chamados de *Sistemas de Tempo Real*. Exemplos de sistemas reativos de tempo real são encontrados no campo automotivo (controle de combustão), nas indústrias de manufatura (controladores de robôs), e nas indústrias de telecomunicações (telefones celulares).

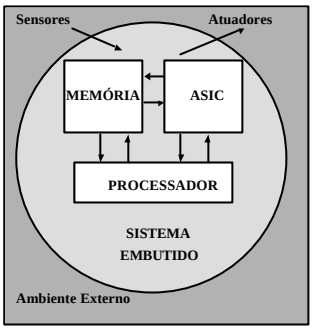


Figura 1.1 - Os Principais componentes de um sistema embutido [1]

Cada um dos três componentes básicos de um sistema embutido é desenvolvido em tecnologia de integração em larga escala (VLSI - "Very Large Scale Integration"). Essa tecnologia permite agregar grande quantidade de operações em circuitos de dimensões microscópicas. Como o uso de VLSI resulta em circuitos bastante complexos, essa tecnologia requer o uso de ferramentas de projeto. Tais ferramentas não somente garantem que a complexidade do circuito seja tratável, mas que o tempo de projeto seja aceitável. A redução do tempo de projeto é especialmente importante para os ASICs. Por não serem produzidos em volumes tão grandes quanto os circuitos de propósitos gerais, os ASICs apresentam dificuldade de amortizar o custo de desenvolvimento. Assim, a redução do tempo de projeto de ASICs é fundamental para sua viabilidade econômica. Isso requer que as ferramentas de projeto sejam eficientes.

Essas ferramentas, que permitem o projeto auxiliado por computador ou CAD ("Computer Aided Design"), passaram a ser vistas como imprescindíveis para que os produtos tenham seus desempenhos garantidos, com custos de desenvolvimento e produção reduzidos e rápida inserção no mercado ("time to market"). Tais ferramentas permitem a automação do projeto eletrônico ou EDA ("Eletronic Design Automation"), solucionando problemas complexos e automatizando trabalhos repetitivos e exaustivos, as ferramentas de projeto tornam tratável a tecnologia VLSI, diminuem a quantidade de erros humanos e reduzem o tempo de projeto.

As técnicas a serem apresentadas neste trabalho pretendem contribuir para a melhoria de uma classe particular de ferramentas para o projeto automatizado de ASICs: as ferramentas de *Síntese de Alto Nível* ou HLS ("High-Level Synthesis"), como descrito na Seção 1.1.

1.1 - Contexto

Quando se refere a etapas de projeto de Sistemas Embutidos, considera-se a metodologia composta de essencialmente cinco etapas (Requisitos do Cliente, Especificação, Arquitetura, Componentes e Integração) [3], visto que está se tratando de um sistema composto por hardware e software. No caso particular do projeto de ASIC's - que são componentes eletrônicos - há tradicionalmente 4 etapas na sua criação: Projeto, Fabricação, Testes e Encapsulamento [1]. Este trabalho relaciona-se com a etapa de Projeto, a qual é comumente subdividida em 3 tarefas principais, que são:

?? *Conceitualização e Modelagem*: consistem em capturar a idéia em um modelo, o qual representa a função que o circuito realiza.

?? *Síntese e Otimização*: A síntese consiste em refinar o modelo, fazendo com que este, que partiu de uma forma abstrata, seja agora progressivamente detalhado, com todas as características necessárias para sua fabricação. Uma meta do projetista é a otimização de alguns parâmetros de projeto, que servem de métrica para a qualidade do circuito final.

?? *Validação*: consiste em verificar a consistência dos modelos utilizados durante o projeto.

Modelos de circuitos são usados para representar idéias. A modelagem é realizada através de representação textual usando Linguagens de Descrição de Hardware ou HDLs ("Hardware Description Languages"), e através de representações gráficas tais como grafos de fluxo de dados, diagramas esquemáticos e de leiaute. Um exemplo de HDL é VHDL [2].

A Síntese do circuito é o segundo processo criativo [1]. O primeiro se dá quando os projetistas pensam como conceitualizar um circuito e esboçar um primeiro modelo. A

principal meta da síntese do circuito é gerar um modelo detalhado de um circuito, tanto quanto o seu layout, que será utilizado para a fabricação do componente (ou "chip"). Esse objetivo é alcançado através de um processo de refinamento progressivo, durante o qual o modelo abstrato original vai sendo iterativamente detalhado pelo projetista. Como a síntese propicia o refinamento do modelo, mais informações são necessárias, considerando a tecnologia e o estilo de implementação de projetos desejados. Conseqüentemente, um modelo funcional de um circuito pode ser razoavelmente independente dos detalhes de implementação, enquanto que o "layout" considera todos os detalhes específicos da tecnologia utilizada (por exemplo, as dimensões dos dispositivos semicondutores).

A meta da síntese não é simplesmente obter automaticamente um circuito que satisfaça as especificações. Quer-se obter automaticamente um circuito que satisfaça os requisitos da especificação e tenha uma qualidade competitiva frente a um circuito projetado manualmente. Por essa razão, a otimização do circuito é geralmente combinada com a síntese. Para se avaliar a qualidade de um circuito, algumas figuras de mérito são levantadas e utilizadas como métricas de qualidade do circuito sintetizado. Algumas figuras de mérito são compulsórias, pois são ditadas pela especificação. Estas são conhecidas como *restrições* de projeto. Uma restrição pode ser assim classificada:

?? *Restrição de precedência*: requer que um evento seja precedido por outro.
?? *Restrição de recurso*: limita o número de recursos alocados para uma tarefa.
?? *Restrição de tempo*: especifica o tempo (mínimo, máximo ou exato) para o término de uma tarefa.

Outras figuras de mérito não são obrigatórias mas desejáveis, pois traduzem a melhor qualidade do circuito sintetizado. Estas são conhecidas como *objetivos* da síntese. Algumas figuras de mérito tradicionalmente utilizadas são:

?? *Latência*: tempo que o circuito leva para responder a um estímulo.
?? *Throughput*: quantidade de informação processada na unidade de tempo.
?? *Área*: área ocupada pelo circuito digital quando integrado numa pastilha.
?? *Potência*: quantidade de energia consumida pelo circuito na unidade de tempo.
?? *Testabilidade*: a facilidade em se testar o "chip" após sua confecção.

Em suma, a síntese deve garantir que *todas* as restrições sejam obedecidas e que pelo menos alguns objetivos sejam otimizados.

A modelagem de sistemas digitais pode ser feita sob diferentes *visões* e em *níveis* distintos de abstração. A Figura 1.2 ilustra esquematicamente esta sucessão de etapas, envolvendo diferentes níveis de abstração. Trata-se de um diagrama clássico conhecido como diagrama Y, proposto por Gajski e Kuhn [1], que é descrito a seguir.

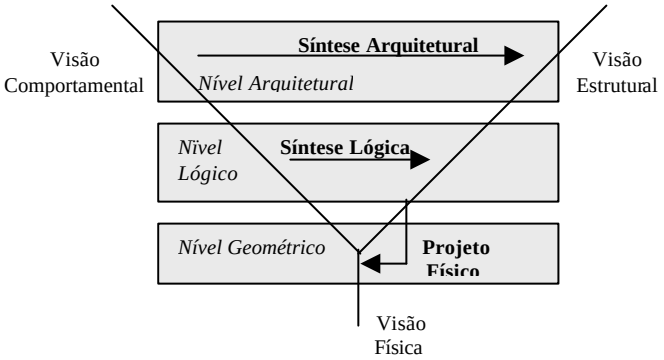


Figura 1.2 - Diagrama Y

As diferentes visões consideradas na elaboração de um modelo são classificadas como comportamental, estrutural e física [1]. A *visão comportamental* descreve a função do circuito independente da sua implementação. A *visão estrutural* descreve um modelo como uma interconexão de componentes, e a *visão física* relaciona os objetos físicos (por exemplo os transistores) de um projeto.

No nível arquitetural, a descrição do comportamento de um sistema é essencialmente uma descrição algorítmica, enquanto que sua descrição estrutural é expressa como uma interconexão de unidades funcionais e registradores, tradicionalmente conhecida como descrição RT ("Register-Transfer").

Por exemplo, no nível arquitetural, a visão comportamental de um circuito é o conjunto de operações e suas dependências, enquanto que a visão estrutural é a interconexão dos componentes que implementam aquelas operações.

Por outro lado, no nível lógico, a visão comportamental do circuito pode ser dada através do diagrama de transição de estados, enquanto que sua visão estrutural é a interconexão de portas lógicas e "flip-flops" que implementam aquele diagrama. Para ilustrar o conceito descrito, utilizamos o exemplo de um processador simples que é descrito em HDL, no seu nível arquitetural.

A Figura 1.3 mostra como são focados os objetivos de cada nível para este exemplo, desde especificação no nível lógico até detalhes para a manufatura do circuito.

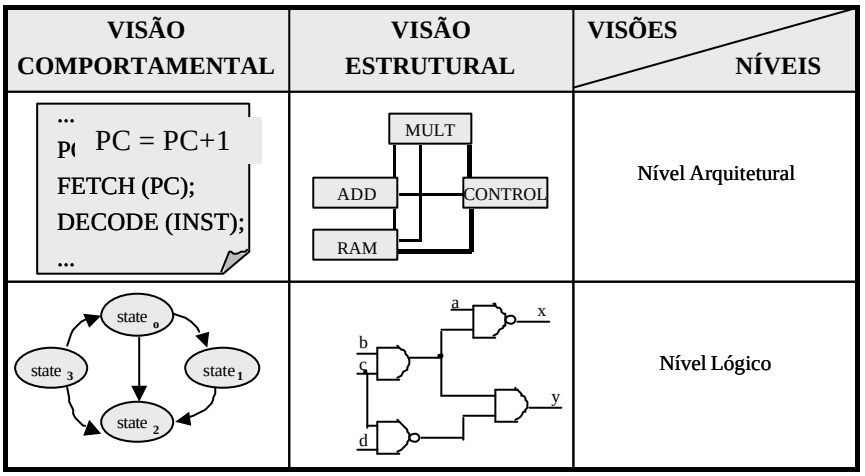


Figura 1.3 - Modelagem de sistemas digitais: visões e níveis de abstração [1]

A Síntese de Alto Nível (HLS) ou Síntese Comportamental foi definida como o processo de obtenção automática da estrutura arquitetural do circuito (registradores, somadores, multiplicadores, etc.) a partir de uma especificação algorítmica de seu comportamento [4]. O comportamento e a estrutura de um circuito são comumente descritos através de uma HDL. A Síntese Lógica consiste na transformação de uma descrição comportamental no nível lógico (função Booleana ou diagrama de transição entre estados) na estrutura de um circuito lógico (portas lógicas e flip-flops). Por fim, a Síntese Física traduz o circuito

lógico no "layout" de um circuito eletrônico, de forma que interconexões de transistores, resistores e capacitores do circuito eletrônico definam os padrões geométricos ("layout") do circuito integrado a ser fabricado.

A Figura 1.4 ilustra esquematicamente o resultado da HLS. A estrutura de um sistema digital no nível arquitetural consiste de uma unidade operativa ("datapath") e uma unidade de controle, supondo que esteja implementando um algoritmo onde são realizadas sete operações (a, b, c, d, e, f, g).

O datapath consiste na interconexão de componentes de três tipos principais:

?? *Unidades funcionais*: somadores, multiplicadores, ALUs, etc;

?? *Elementos de armazenamento* : memórias, registradores;

?? *Elementos de conexão*: barramentos, multiplexadores, etc.

Em resumo, o resultado da HLS consiste em uma descrição RT da estrutura do "datapath" e em uma descrição do comportamento da unidade de controle, através de uma máquina de estados finitos simbólica. A síntese da estrutura da unidade de controle a partir da máquina de estados é objeto da Síntese Seqüencial e da Síntese Lógica e está fora do âmbito deste trabalho.

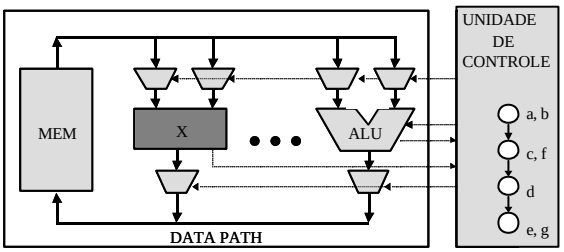


Figura 1.4 - Estrutura no nível arquitetural

Dada uma descrição comportamental de um circuito, ela será capturada em modelos capazes de representar as funções do circuito, as restrições e o objetivo de projeto. É a partir dessa modelagem que se realiza o processo de HLS. É um processo complexo, que precisa ser dividido em etapas, a saber:

?? *Seleção*: define quais os tipos de recursos necessários para executar as operações;

?? *Alocação*: define quantos elementos de cada tipo de recurso são necessários.

?? *Escalonamento*: determina qual a seqüência de execução das operações, ou seja, quando cada uma delas será realizada.

?? *Ligação*: associa cada operação com o recurso onde será executada.

1.2 Motivação

Um exemplo de metodologia para o processo de síntese é ilustrado na Figura 1.5, adaptada do original [5]. Partindo da descrição comportamental, a seleção e a alocação precedem o escalonamento, impondo-lhe restrições de recursos. Isso caracteriza o escalonamento sob restrições de recursos, objetivo deste trabalho. Note que a metodologia resulta em um processo iterativo, em que soluções são geradas até que todas as restrições sejam satisfeitas.

Quando a estrutura da solução satisfaz todas as restrições, passa-se à síntese em outros níveis de abstração (lógica, layout).

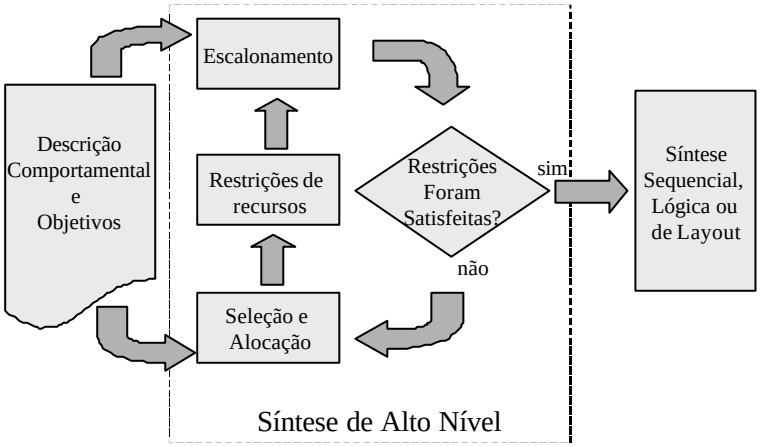


Figura 1.5 – Um fluxo de projeto para as ferramentas de síntese

A realização de uma operação requer que os valores dos operandos sejam lidos dos registradores fonte, que sejam roteados para uma unidade funcional e que o resultado seja roteado para um registrador destino. Isso caracteriza a noção de transferência entre

registradores, ou RT (“Register Transfer”). Neste trabalho assume-se que o roteamento de operandos e de resultados seja feito através de barramentos.

Pode haver vários recursos disponíveis para a realização simultânea de operações distintas, mas estas só se realizarão simultaneamente de fato se houverem barramentos disponíveis para o transporte dos seus respectivos dados (operandos e resultados). Se a quantidade de dados a ser transportada ultrapassa a capacidade de tráfego pelos barramentos e não se pode ou não se quer aumentar o seu número, é necessário definir quais operações terão prioridade para que sejam os seus operandos os eleitos para transitarem pelos barramentos disponíveis.

Em outras palavras, a limitação de recursos do tipo barramento impõe restrições ao escalonamento das operações. Ora, este é exatamente o objeto deste trabalho.

A principal motivação para a pesquisa descrita nesta dissertação é amparar fluxos de projeto tais como o da Figura 1.5, onde a alocação precede o escalonamento. A alocação define os componentes do “datapath”, entre eles o número de barramentos. A ferramenta de alocação busca minimizar o número de barramentos para diminuir a área gasta com interconexões no circuito integrado, diminuindo o seu custo. Conseqüentemente, a ferramenta usada para escalonamento deve suportar um número pré-fixado de barramentos como restrição de recursos.

Ao suportar restrições de barramentos impostas ao escalonamento, o fluxo de projeto é capaz de encontrar o número mínimo de barramentos para se atingir a mínima latência, para um dado número de UFs.

Um maior número de barramentos implica em um maior número de conexões, o que leva a uma maior área ocupada em silício e/ou a uma maior dificuldade de estabelecer conexões no futuro circuito integrado. Como o barramento insere capacitâncias e resistências no circuito elétrico, um maior número de barramentos implica em maior potência consumida pelo circuito para carregar

e descarregar as capacitâncias. Ora, muitas aplicações em sistemas embutidos requerem baixo consumo de potência. Em resumo: a minimização do número de barramentos resulta em uma melhor qualidade do futuro circuito integrado.

1.3 Organização

Esta dissertação é organizada da seguinte forma:

O Capítulo 2 introduz os modelos utilizados na representação de comportamento e estrutura. Em seguida, mostra-se a modelagem do problema de escalonamento e de sua solução. Em especial apresenta-se a modelagem das restrições impostas pelos barramentos. Ao final do capítulo, faz-se uma análise de trabalhos correlatos.

O Capítulo 3 mostra como a modelagem de tais restrições se integra ao escalonamento. Um algoritmo de escalonamento é adotado e estendido para manipular as restrições impostas pela alocação de barramentos.

A implementação do algoritmo de escalonamento estendido é discutida no Capítulo 4. Neste capítulo, descrevem-se os experimentos realizados com o protótipo e os respectivos resultados.

O Capítulo 5 traz as conclusões e as perspectivas de continuidade deste trabalho.

As referências bibliográficas encontram-se no Capítulo 6.

As características dos exemplos utilizados nos experimentos são apresentadas nos anexos.

O trabalho de pesquisa aqui descrito foi realizado no âmbito do projeto “DESERT – Desenvolvimento de Ferramentas para Sistemas Embutidos de Tempo Real”, dentro do Grupo de Trabalho “OASIS – Modelagem, Otimização e Síntese de Arquiteturas e Sistemas Digitais”.

O Projeto DESERT é fomentado pelo CNPq. Em especial, o trabalho de pesquisa desenvolvido nesta dissertação foi parcialmente suportado por bolsa do CNPq, no âmbito do Plano Nacional de Microeletrônica.

2 Modelagem

Neste capítulo são formalizadas as noções que permitem a modelagem de um sistema digital no nível arquitetural. A Seção 2.1 define os modelos para o comportamento do sistema digital, para sua estrutura e para as restrições a ele impostas.

Este trabalho busca resolver um problema de Síntese de Alto Nível: o escalonamento sob restrição de recursos. Muitos trabalhos correlatos [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16] limitam-se ao tratamento de recursos do tipo unidade funcional (UF). Este trabalho, além de tratar restrições devidas a UFs, aborda também as restrições de recursos do tipo barramento.

Nas Seções 2.2 e 2.3, este problema será primeiramente introduzido através de exemplos e, em seguida, definido formalmente.

Vale lembrar que, embora haja diferentes tipos e implementações de barramentos, no nível arquitetural todos têm essencialmente o mesmo efeito de limitar o paralelismo das operações a serem executadas no "datapath". Portanto, este trabalho limita-se a modelar as restrições impostas pelos barramentos à execução simultânea de operações.

2.1 - Definições Básicas

A modelagem do comportamento do sistema, da estrutura de sua unidade operativa e do comportamento de sua unidade de controle é feita através de três grafos distintos, descritos a seguir.

Definição 2.1 - Um *grafo polar de fluxo de dados* DFG(V,E) é um grafo orientado onde cada vértice v_i ? V representa uma operação, e onde cada aresta (v_i, v_j) ? E representa uma dependência de dados entre as operações v_i e v_j. Os pólos são os vértices v_0 e v_n, denominados fonte e sumidouro.

Note que cada aresta (v_i, v_j) do DFG representa o transporte de um valor produzido pela operação v_i e consumido pela operação v_j, ou seja, a aresta (v_i, v_j) transporta o valor de um operando de v_j.

Definição 2.2 - Um *grafo polar da máquina de estados* SMG(S,T) é um grafo orientado onde cada vértice s_i ? S representa um estado, e onde cada aresta (t_i, t_j) ? T representa uma transição entre os estados s_i e s_j. Os pólos são os vértices s_0 e s_n, denominados fonte e sumidouro.

Definição 2.3 - Um *grafo polar do datapath* DPG (C, W) é um grafo orientado onde cada vértice c_i ? C representa um componente, e onde cada aresta (w_i, w_j) ? W representa uma interconexão entre os componentes c_i e c_j. Os pólos são os vértices c_0 e c_n, denominados fonte e sumidouro.

Assume-se que cada aresta emergente de um dado vértice do DFG receba um rótulo único e esteja associada ao valor transportado por aquela aresta. Em outras palavras, o rótulo é um nome simbólico para o valor efetivamente calculado pela operação em tempo de execução, conforme formalizado a seguir.

Definição 2.4 – Seja uma operação v ? V, seus predecessores u e w e seu sucessor z. Os *operandos* da operação v, denotados por $v.operando_1$ e $v.operando_2$ são os rótulos das arestas (u,v) e (w,v), respectivamente. O resultado da operação v, denotado por v.resultado, é o rótulo da aresta (v,z).

A execução propriamente dita de uma operação ocorre em unidades funcionais (UFs) tais como ALUs, multiplicadores, etc. O roteamento dos dados (operandos e resultados) para viabilizar a execução, requer recursos do tipo barramento.

Conforme caracterizado anteriormente, existe uma associação entre as operações e os tipos que as realizam, como por exemplo operações de adição, multiplicação, de acesso à

memória para leitura de dados, de acesso à memória para escrita de dados, transferência de dados, etc. Para a execução de operações, é preciso alocar um número finito de *recursos físicos*, doravante denominados *recursos*, por simplicidade. Às unidades funcionais estão associados tipos, tais como ALU, multiplicador, etc. Estas noções estão formalizadas abaixo.

Definição 2.5 - Um *vetor de restrição de recursos* $\boldsymbol{a}$ é um vetor onde cada componente a_k representa o número de unidades funcionais disponíveis de um determinado tipo k_? _{1,2,...n}.

Definição 2.6 - Uma função ? : V? {1,2,...n} é uma função que mapeia cada operação para um tipo de recurso k_? _{1,2,...n} onde será executada.

Depois de consumir operandos, cada operação produz um novo valor como resultado. Essa produção necessita de um certo tempo chamado *atraso de execução,* conforme formalizado a seguir.

Definição 2.7 - O *atraso de execução* d_i de uma operação é um valor correspondente ao número de ciclos necessários para completar a execução de uma operação v_i em um recurso do tipo ? (v_i).

A Figura 2.1 ilustra o atraso de execução de operações, onde os ciclos de relógio são representados esquematicamente por linhas tracejadas horizontais. Nela são mostradas três operações de tipos distintos: uma como de leitura de memória ou "load" (LD), uma multiplicação e uma adição. Observa-se que tanto a operação load "A", quanto a adição "B" necessitam de um ciclo de relógio para completarem suas execuções ($d_A = d_B = 1$), enquanto a multiplicação "C" necessita de dois ciclos de relógio ($d_C = 2$).

No nível arquitetural, costuma-se supor que o atraso de propagação dos dados no barramento é nulo. Essa hipótese justifica-se aqui porque o atraso no barramento é desprezível frente ao atraso de execução das UFs ou porque aquele atraso é agregado ao das

UFs para efeito de modelagem. Por isso, neste trabalho, adota-se a hipótese de atraso nulo nos barramentos.

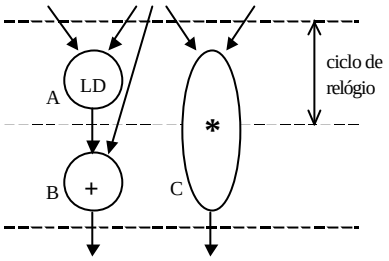


Figura 2.1 - Exemplo de atraso na execução de operações

Escalonar um DFG significa dispor as operações ao longo do tempo, respeitando as restrições de precedência e as restrições de recursos, conforme formalizado a seguir:

Definição 2.8 - Uma função denominada *escalonamento* ? : V? S é uma função que mapeia cada vértice v_i do DFG para um estado s_k = ? (vi) do SMG, tal que:

?? ? (v_i, v_j) ? E : (s_k = ? (v_i) e s_r = ? (v_j)) ? r ? $k + d_i$ (restrição de precedência), e

?? ?{v_i : ? (v_i) = p e k ? $m < k + d_i$ } ? ? a_p, para cada tipo de recurso p = 1, 2, ...n e para cada estado s_m, com m = 1, 2, ...n (restrição de recursos).

Alguns algoritmos utilizados neste trabalho baseiam-se na noção de caminhos em grafos, como formalizado a seguir.

Definição 2.9 - Um *caminho* em um grafo orientado G(V,E), com início no vértice v_0 e término no vértice v_n, é a seqüência (v_0, v_1, v_2, ...v_n) de vértices tal que (v_{i-1}, v_i) ? E , e i_? {1, 2, ...n}.

Definição 2.10 - Dado um grafo orientado G(V,E) e dois vértices arbitrários v_i e v_j ? V, o vértice v_i alcança v_j através de p, escrito v_i ? v_j, se há um caminho p de v_i até v_j.

Muitas vezes o caminho não necessita ser nomeado, escrito v_i* ? v_j. Notamos que este caminho poderá ser trivial caso v_i = v_j.

Definição 2.11 - O *caminho mais longo* (“longest path”) entre dois vértices arbitrários v_i e v_j ? V em um grafo orientado G (V,E), é o caminho com o maior número de vértices tal que v_i ? v_j.

Definição 2.12 - Dado um grafo orientado G(V,E) e dois vértices arbitrários v_i e v_j ? V, tais que v_i ? v_j, a distância entre os vértices v_i *até* v_j denotada por ? (v_i, v_j) é igual ao número de arestas no caminho *p*.

O escalonamento define o número de ciclos de relógio em que o algoritmo de uma descrição comportamental é executado, resultando na noção de latência, que é formalizada abaixo.

Definição 2.13 - A latência ? do SMG é igual ao número de estados pertencentes ao caminho mais longo entre o vértice fonte v_0 e o vértice sumidouro v_n.

Durante o escalonamento as operações são associadas a estados. As operações passíveis de escalonamento em um dado estado são aquelas que ainda não foram escalonadas e cujos predecessores foram todos escalonados, há tempo suficientemente longo para acomodar seus atrasos de execução, conforme formalizado a seguir.

Definição 2.14 - Dado um SMG e um estado arbitrário s_k ? S, o conjunto das *operações prontas* em s_k, denotado por A_k, é o conjunto de todas as operações v_j, tais que:
?? a operação v_j não foi escalonada em s_k ou em algum estado s_m tal que s_m ? s_k;
?? para todo predecessor imediato v_i *de* v_j escalonado em um estado arbitrário s_p, vale a desigualdade ? (s_p, s_k) ? d_j.

Nem todas as operações podem ser executadas em um único ciclo. Por isso, se uma operação v_j, com $d_j > 1$, for executada em um estado s_m, ela não estará finalizada no estado s_k, caso a distância entre s_m e sk for menor do que d_j. Essa noção é formalizada a seguir.

Definição 2.15 - Dado um SMG e um estado arbitrário s_k ? S, o conjunto das *operações pendentes* em s_k, denotado por U_k, é o conjunto de todas as operações v_j, tais que:

?? a operação v_j já foi escalonada num estado arbitrário s_m tal que s_m ? s_k;*

?? ? $(s_m, s_k) < d_j$.

2.2 - O Problema de Escalonamento

Informalmente, o escalonamento consiste em encontrar um ordenamento de operações ao longo do tempo, obedecendo às restrições de precedência (produção e consumo de dados) e restrições de recursos (apenas uma operação pode ocupar um recurso em um determinado instante), de forma a minimizar o tempo total de execução.

Antes de formalizar o problema de escalonamento, ele será ilustrado através de exemplos. A Seção 2.2.1 mostra a especificação de um problema-exemplo. A Seção 2.2.2 mostra um exemplo de escalonamento.

2.2.1 - Especificação do Problema-Exemplo

A Figura 2.2 mostra um trecho de descrição comportamental contendo duas multiplicações, uma operação lógico-aritmética (+), cinco operações de acesso à memória para leitura de dados e 2 operações de acesso à memória para armazenagem de dados, caracterizando assim os quatro tipos de recursos: multiplicador (MUL), unidade de lógica aritmética (ALU), porta de escrita de dados em memória (*store* ou STR) e porta de leitura de dados de memória (*load* ou LD). Alguns valores produzidos são consumidos por outras operações, dando origem às *restrições de precedência*. Por exemplo, o valor "c" (na linha G) precisa ser calculado antes de "e" (na linha I).

A execução de uma operação em um recurso requer um intervalo de tempo, conhecido como *atraso de execução* (ver Definição 2.7), que é expresso em ciclos de relógio. Para este exemplo, assume-se que o atraso de execução é de um ciclo de relógio para todas as

operações. No caso particular de operações “load” e “store”, o atraso de execução é o intervalo de tempo para realizar um acesso de leitura ou de escrita na memória respectivamente.

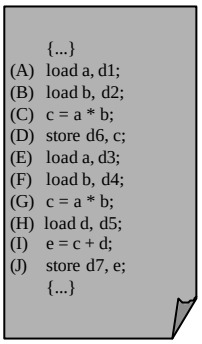


Figura 2.2 - Trecho de uma descrição comportamental

A Figura 2.3 mostra um DFG obtido na descrição comportamental da Figura 2.2, onde círculos representam as operações e arestas com linhas cheias representam dependência de dados. Os pólos do DFG, denominados de fonte (“source”) e sumidouro (“sink”), servem apenas para a identificação das entradas e saídas primárias e têm atraso de execução nulo.

As arestas pontilhadas com origem no vértice fonte representam os valores das entradas primárias. Similarmente, as arestas pontilhadas incidentes no vértice sumidouro representam os valores das saídas primárias. Por simplicidade, considerando o fato de que as entradas primárias estão disponíveis o tempo todo, algumas arestas que transportam os valores das entradas primárias serão doravante ocultadas nas ilustrações. Este é o modelo de DFG utilizado neste trabalho.

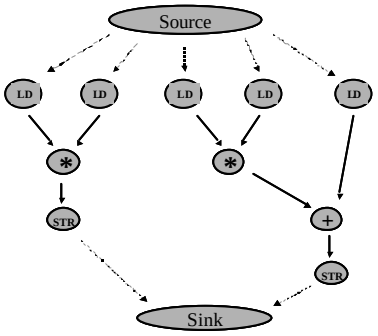


Figura 2.3 - DFG obtido a partir da descrição comportamental da Figura 2.2

2.2.2 - Exemplos de escalonamentos

Dados o DFG da Figura 2.3 e os atrasos de execução mencionados na seção anterior, vamos associar as operações a diferentes instantes de tempo, respeitando as restrições de precedência e de recursos.

Uma primeira solução pode ser construída assumindo-se um único multiplicador, uma única ALU e uma memória com duas portas de leitura e uma de escrita, conforme ilustra a Figura 2.4. Nela, as linhas horizontais delimitam diferentes ciclos de relógio. Cada ciclo representa a duração de um estado da máquina.

Observa-se que cada operação é associada a um tempo inicial de execução. O instante de tempo em que uma operação termina sua execução (que é o tempo inicial de seu sucessor) é obtido pela soma de seu tempo inicial e de seu atraso de execução. Observa-se que são necessários 5 estados para acomodar todas as operações, portanto para esta solução, ?=5. Entretanto, soluções diferentes podem ser encontradas alterando-se a quantidade de recursos disponíveis.

Como as operações A, B, C, D e E requerem um mesmo recurso do tipo porta de leitura de dados da memória, como será explicado no Capítulo 3, um critério de prioridade precisa ser adotado para se escolher entre essas operações, onde se definirá quais vão ocupar as duas portas de leitura de dados da memória disponíveis, no primeiro estado (ciclo 1).

Assume-se, sem perda de generalidade, que as operações A e B tenham sido as escolhidas.

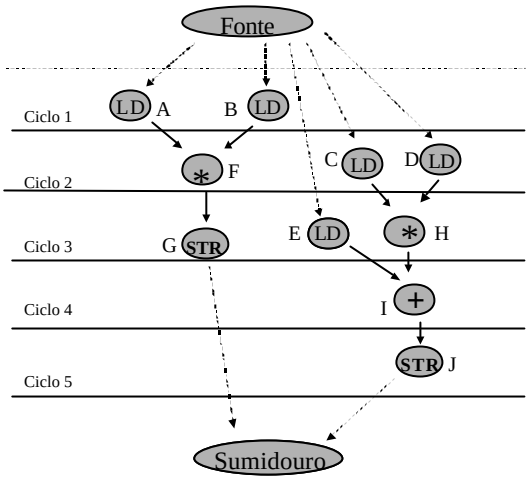


Figura 2.4 - Escalonamento do DFG

Como estas possuem um atraso unitário de execução, ambas as portas de leitura da memória estarão ocupadas durante o estado s_0. A partir do estado s_1, a operação F torna-se apta a consumir os valores produzidos por A e B no estado s_0. De forma similar, as demais operações são associadas a outros estados, até que todas as operações tenham sido escalonadas.

2.2.3 - Formalização do Problema

Formalmente o escalonamento sob restrições de recursos pode ser formulado como um problema de otimização combinatória, como segue:

Problema 2.1 - Dado um grafo de fluxo de dados DFG (V, E) e um vetor de restrição **a**, encontre um escalonamento ? que minimize a latência ?.
Esse é o problema mais freqüentemente abordado pelas técnicas de escalonamento reportadas na literatura [5, 6, 7, 8, 9, 12, 13, 14, 15, 16, 18 e 19].

Entretanto, uma técnica mais pragmática deveria admitir não somente restrições impostas pelas unidades funcionais, mas também aquelas impostas por um número pré-fixado de barramentos. Isso dá origem a um problema mais geral, assim enunciado:

Problema 2.2 - Dado um grafo de fluxo de dados DFG (V, E), um vetor de restrição **a** e um número de barramentos **N**, encontre um escalonamento ? que minimize a latência ?.

O escalonamento sob restrição de recursos é um problema bem conhecido de otimização combinatória [3], pertencente à classe de problemas intratáveis (NP completos). É um problema clássico nas áreas de Compiladores e CAD (Projeto Auxiliado por Computador).

Um dos objetivos deste trabalho é generalizar um método de solução do Problema 2.1, de forma a resolver também o Problema 2.2. Um outro objetivo é comparar as soluções obtidas para ambos os problemas de forma a quantificar o impacto das restrições impostas pelos barramentos.

2.3 - Restrições Impostas pelos Barramentos

As operações são executadas em unidades funcionais, as quais, por serem circuitos combinacionais, não possuem capacidade de armazenamento.

Em conseqüência, o valor produzido por uma operação deve ser armazenado em um registrador, até ser consumido por outra operação. Para esse armazenamento, há a necessidade de disponibilizar registradores em número suficiente, para neles armazenar valores até que sejam consumidos.

Uma outra possibilidade seria a de armazenar os valores produzidos em uma memória até que sejam requisitados para serem consumidos.

O papel dos barramentos é justamente o de transportar os dados entre registradores, memória e unidades funcionais.

A quantidade de barramentos existente no "datapath" limita o paralelismo potencial entre as operações. O caso extremo seria a existência de um único barramento. Neste caso, não haveria paralelismo algum entre operações, pois uma única operação teria que utilizá-lo três vezes para completar sua execução (duas para receber os operandos e uma terceira para armazenar o resultado).

Daí a necessidade de modelar a quantidade de barramentos existentes no datapath e monitorar quais estão em uso e quando serão liberados.

Definição 2.16 - Um *vetor de ocupação de barramentos* é um vetor **b** = (b_1, b_2, ..., b_k, ... b_N) cujo número de elementos N é o número total de barramentos no "datapath" e onde cada componente b_k é tal que:

?? Se b_k = nenhum, então o k-ésimo barramento está livre;

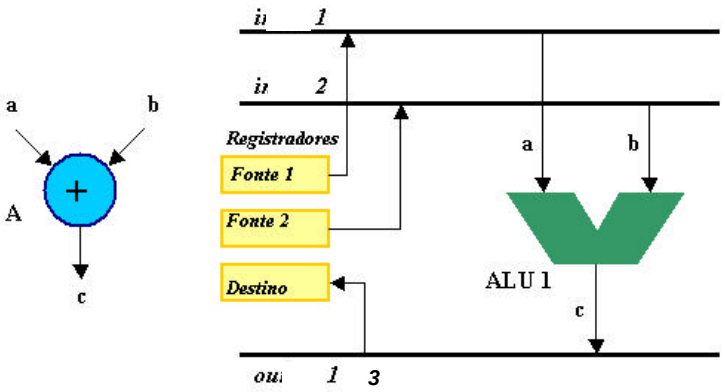
?? Se b_k ? nenhum, então o k-ésimo barramento está ocupado.

No caso de um barramento estar ocupado, ao componente b_k é atribuído o valor do operando ou resultado por ele transportado.

Conquanto um barramento só pode transportar um valor por vez, se um mesmo operando é consumido por operações diferentes escalonadas em um mesmo estado do SMG, ele pode ser transportado através de um mesmo barramento para unidades funcionais distintas.

A seguir, o conceito de restrições impostas pelos barramentos será primeiramente introduzido através de um exemplo e, depois, formalizado.

A noção de transferência entre registradores (RT), essencial à modelagem de restrições de barramentos, é ilustrada através da Figura 2.5.



(a) (b)

Figura 2.5 – Relação entre uma operação e uma RT

A Figura 2.5a mostra uma operação A que consome os operandos **a** e **b** e produz o resultado **c**. A Figura 2.5b ilustra parte da estrutura de um datapath onde a operação A será realizada. A execução da operação ocorrerá na ALU1.

Suponha que os valores dos operandos **a** e **b** estejam armazenados nos registradores fonte1 e fonte2 e que o resultado **c** será armazenado no registrador destino. Para rotear tais valores entre registradores e ALU são utilizados 2 barramentos conectados às entradas da ALU (bus1 e bus2) e um barramento conectado à saída da ALU (bus3).

Uma RT representa o seguinte conjunto de ações:

?? a leitura dos registradores-fonte, que armazenam os valores dos operandos (*a* e *b*);

?? seu transporte através de barramentos (bus1 e bus2), até a entrada da ALU;

?? o transporte do resultado através de um barramento (bus3);

?? a escrita do resultado (c) no registrador destino.

A noção de transferência entre registradores pode ser formalizada como segue.

Definição 2.17 - Uma RT é uma *t-upla* (operando$_1$, inbus$_1$, operando$_2$, inbus$_2$, resultado, outbus), onde:

?? *operando$_1$* é o primeiro operando-fonte, armazenado previamente em um registrador;

?? *inbus*$_1$ é o barramento que transporta *operando*$_1$ de um registrador-fonte para uma das entradas de uma UF;

?? *operando*$_2$ é o segundo operando-fonte, armazenado previamente em um registrador;

?? *inbus*$_2$ é o barramento que transporta *operando*$_2$ de um registrador-fonte para uma das entradas de uma UF;

?? *resultado* é o resultado da operação realizada na unidade funcional;

?? *outbus* é o barramento que transporta o resultado da saída de uma UF até o registrador destino.

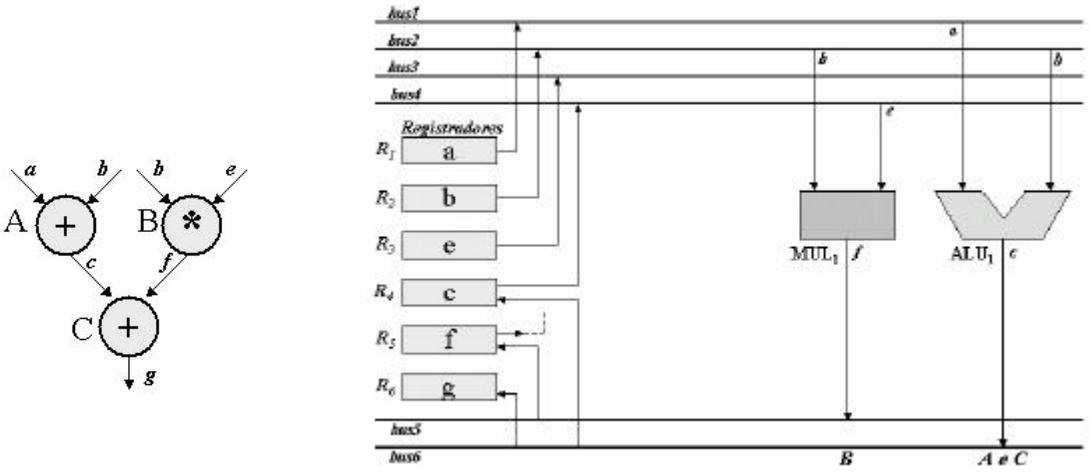
Assim, para o exemplo da Figura 2.5:
RT_A={a, bus1, b, bus2, c, bus3}.

Para denotar o valor de um atributo ***x*** de uma RT, adotar-se-á a notação RT.x. Por exemplo, o valor do elemento "resultado" de uma RT é denotado por RT.resultado.

Uma RT descreve um caminho no "datapath" que resulta na execução de uma operação. Nessa modelagem, assume-se que o caminho determinado pela RT esteja ativo durante todo o tempo em que a respectiva unidade funcional estiver ativa, ou seja, durante o seu atraso de execução. Isso significa que, durante esse intervalo, o transporte de dados através de barramentos pertencentes àquele caminho só é permitido se não interferir com a operação implementada através daquela RT.

Ora, durante o escalonamento pode ocorrer que, em um determinado estado, duas operações estejam prontas para executar, pois seus operandos estão disponíveis para consumo, mas talvez não haja um caminho livre no "datapath" que permita a propagação simultânea dos valores de operandos e resultados sem interferência mútua. A garantia de que duas operações não interfiram mutuamente quando escalonadas em um mesmo estado será avaliada através da noção de compatibilidade entre RTs. Essa noção será introduzida através do exemplo da Figura 2.6.

A Figura 2.6a mostra um DFG com três operações A, B e C a serem realizadas no *datapath* da Figura 2.6b.



a) b)

Figura 2.6 – Exemplo ilustrativo da compatibilidade entre RTs

Suponha que as operações A e B sejam realizadas através das seguintes RTs, respectivamente: RT_1=(a, bus1, b, bus2, c, bus5) e RT_2=(e, bus3, b, bus2, f, bus6).

Como uma operação pode ser realizada por RTs distintas, a operação B poderia também ser realizada através das seguintes RTs: RT_3=(e, bus3, b, bus2, f, bus5) e RT_4=(b, bus2, e, bus3, f, bus6), por exemplo.

Deve-se notar que, mesmo que as operações A e B seja escalonadas no mesmo estado, as transferências RT_1 e RT_2 não resultam em conflito de valores no barramento. Embora $RT_1.inbus_2$ = $RT_2.inbus_2$ = bus2, o valor transportado pelo bus2 é o mesmo (b) em ambas RTs. Diz-se que RT_1 e RT_2 são *compatíveis*. RT_1 e RT_4 são também compatíveis, pois embora $RT_4.inbus_1$ = $RT_1.inbus_2$ = bus_2, o valor transportado pelo bus_2 é também o mesmo (b).

Por outro lado, nota-se que os resultados de operações distintas não podem ser transportados simultaneamente através de um mesmo barramento, pois seus valores devem ser considerados distintos, uma vez que dependem dos valores dos operandos. Portanto, as

transferências RT_1 e RT_3 resultariam em conflito, pois RT_1.outbus = RT_3.outbus = bus5. Diz-se que RT_1 e RT_3 são *incompatíveis*.

A noção de compatibilidade é formalizada a seguir.

Definição 2.18 - Dadas duas transferências distintas entre registradores, RT_1 e RT_2 diz-se que RT_1 e RT_2 são compatíveis se, e somente se, todas as condições abaixo forem verdadeiras:

?? ? i, j | i,j ? {1, 2}: (RT_1.inbus$_i$? RT_2.inbus$_j$) ? (RT_1.operando$_i$ = RT_2.operando$_j$);

?? ? i | i ? {1, 2}: (RT_1.outbus ? RT_2.inbus$_i$) ? (RT_2.outbus ? RT_1. inbus$_i$);

?? RT_1.outbus ? RT_2.outbus;

Note que a primeira condição da Definição 2.18 garante que um mesmo barramento pode ser utilizado por operações diferentes, somente se ele transportar um operando comum para UFs distintas. A segunda condição garante que um barramento que transporta um resultado não pode transportar um operando, pois se tratam de valores intrinsecamente conflitantes. A terceira condição garante que um barramento que transporta um resultado não pode ser utilizado por RTs distintas.

Esta definição de compatibilidade será capturada pelos algoritmos a serem apresentados no próximo capítulo.

2.4 - Revisão Bibliográfica

2.4.1 - Abordagens do Problema

Uma possível abordagem do problema consiste na utilização de técnicas exatas de resolução. Um exemplo de técnica exata é a Programação Linear Inteira (ILP) [3]. Essa técnica consiste em modelar o escalonamento como um problema de minimização de uma

"função custo" sujeita a restrições representadas por inequações simultâneas. As variáveis de decisão nas inequações estão associadas à atribuição de operações a instantes de tempo. Não se conhecem algoritmos com complexidade polinomial que garantam sempre a obtenção de uma solução ótima para o problema de escalonamento sob restrição de recursos. Por isso, a utilização de algoritmos exatos restringe-se a pequenas instâncias de problema, pois do contrário o tempo computacional necessário seria proibitivo. Em suma, abordagens exatas são, em geral, inadequadas em face da crescente complexidade dos sistemas.

Outra abordagem é a utilização de algoritmos aproximados que, embora não possam sempre garantir a obtenção da solução ótima, buscam um compromisso entre a qualidade da solução e o tempo de execução aceitável. Nesse caso, são utilizados algoritmos polinomiais que escalonam as operações ao longo do tempo de acordo com uma lista de prioridades, obtida a partir de critérios heurísticos. A decisão de escalonar uma operação ou outra é feita com base na maior prioridade. Exemplos de técnicas heurísticas de escalonamento de uso bastante difundido são os algoritmos "List Scheduling" e "Force-Directed Scheduling" [3]. Embora essas heurísticas tenham se mostrado eficientes para várias instâncias do problema de escalonamento, há situações onde a solução encontrada não é de boa qualidade. A deficiência da maioria das abordagens aproximadas é que uma única solução é gerada, não existindo a possibilidade de se explorar soluções alternativas que eventualmente poderiam levar a um menor custo [5].

Diante da inadequação das técnicas exatas e da deficiência da maioria das técnicas aproximadas heurísticas, abordagens baseadas na pesquisa de soluções alternativas têm sido propostas, utilizando algoritmos de busca local, tais como "*tabu search*" [24], algoritmos genéticos [23], etc. A abordagem adotada neste trabalho [5] vai ao encontro destas propostas e consiste na construção e exploração de múltiplas soluções alternativas.

2.4.2 - Trabalhos Correlatos

Para conhecer o estado da arte dentro do tema proposto e situar a contribuição deste trabalho, foram pesquisados temas correlatos na literatura, em especial artigos apresentados em eventos de visibilidade internacional, com ênfase nos últimos cinco anos. Analisando a literatura, observam-se trabalhos correlatos que utilizam técnicas heurísticas [11, 10, 18, 20 e 26] e exatas [14]. Por exemplo, em [18] é apresentada uma técnica heurística que considera o compartilhamento de barramentos através da geração de tabelas de ocupação, preenchidas com base nos resultados do escalonamento.

Em [8] observa-se uma proposta de arquitetura genérica para o "datapath" em que há armazenamento de operandos em diferentes bancos de registradores ou RFs ("Register Files"). Naquela proposta, as RTs estão associadas a um ciclo de relógio, conectando um RF fonte a um RF destino. Cada RT já contém as informações de ligação relativas aos recursos nos quais as ações da descrição de entrada estão mapeadas e as unidades operativas (OPUs) que as executam. As RTs são completamente caracterizadas pela identificação dos recursos usados e de seu modo de operação. Essa arquitetura é mostrada na Figura 2.7.

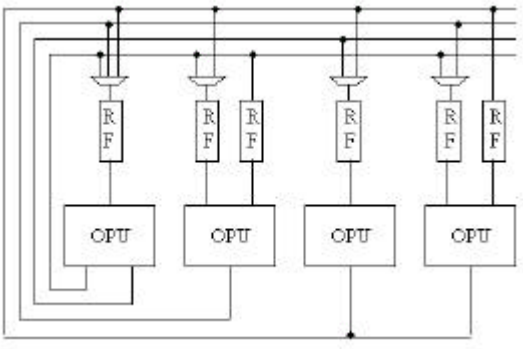


Figura 2.7– Arquitetura genérica do "datapath" proposta em [8]

A arquitetura da Figura 2.7 pressupõe múltiplos bancos de registradores, cuja quantidade é previamente dimensionada e tratada como uma restrição. É diferente da considerada neste e em outros trabalhos [13, 25], que pressupõem um único banco.

Em [8], o número de registradores é pré-fixado antes do escalonamento (restrição de registradores). Nesse aspecto, a abordagem considerada neste trabalho é mais flexível quanto à exploração de soluções alternativas, pois os valores dos operandos ainda não estão mapeados para registradores, a priori. Nesta abordagem utiliza-se os conceitos de alocação de registradores após o escalonamento, ou seja, a quantidade de registradores necessários é conseqüência do escalonamento.

Em resumo, embora este trabalho utilize o conceito de RT similar ao proposto em [8], os trabalhos distinguem-se nos seguintes aspectos:

- Formulação do problema – na abordagem descrita neste trabalho, busca-se construir soluções sob restrições de recursos de conexão e de UFs, enquanto naquele caso [8] consideram-se restrições de registradores;
- Método de solução – o método aqui abordado utiliza um critério de busca local para a exploração de soluções alternativas, enquanto [8] utiliza um método exato;
- Arquitetura-alvo –a arquitetura aqui considerada possui um único banco de registradores, enquanto que [8] pressupõe múltiplos bancos.

Nos últimos anos, parece ter havido uma mudança de foco nas pesquisas em EDA. Ao invés do foco em ferramentas para a otimização de ASICs, há um foco em ferramentas para a otimização de sistemas embutidos como um todo. No que se refere aos meios de conexão, pesquisas recentes concentram-se por exemplo em redes de comunicação [20] ou barramentos para interligação de componentes de sistemas embutidos [19, 21].

Apesar dessa mudança de foco, que aparentemente indica a maturidade das técnicas de otimização de ASICs, há uma carência de resultados quantificando o impacto das restrições de barramentos durante a HLS. Diante dessa carência, este trabalho busca contribuir com alguns resultados experimentais, mostrando o impacto quantitativo da limitação do número de barramentos.

Cabe ressaltar que a carência de trabalhos na literatura sobre o tema específico de escalonamento sob restrições de barramentos é provavelmente um reflexo do fato de este ser

um caso particular do problema mais geral de escalonamento sob restrições de recursos. Entretanto,
essa carência abre a oportunidade de uma contribuição científica: a de avaliar experimentalmente o impacto desse tipo de restrição no problema mais geral.

Essencialmente, este trabalho estende o algoritmo originalmente proposto em [27], e adaptado para HLS em [5, 6, 7], substituindo a noção de operação pela noção de RT. Isso resulta na extensão da abordagem orientada à exploração de soluções alternativas originalmente proposta em [5], como será descrito no próximo capítulo.

3 A Abordagem Explorativa e sua Extensão

Em [6] e [7] foi apresentado um escalonador capaz de ordenar as operações ao longo do tempo, obedecendo as restrições de precedência e as restrições impostas por um número pré-fixado de unidades funcionais.

Neste capítulo, tal escalonador será estendido de forma a torná-lo capaz de também manipular restrições impostas por um número pré-fixado de barramentos. A abordagem adotada é descrita a seguir.

Para aumentar as chances de se encontrar uma boa solução em um tempo computacional aceitável, deve-se explorar diversas soluções, mas sem cair em uma busca exaustiva. A idéia é criar um compromisso entre o tempo de busca e a qualidade da solução obtida. É possível explorar soluções alternativas com a definição de uma codificação de prioridade para determinar em que ordem as operações serão selecionadas para escalonamento [5] (como será ilustrado na Seção 3.2).

Diferentes codificações resultam em diferentes soluções com custos possivelmente distintos. Dessa forma, a otimização do custo é realizada através de monitoração dos custos de diferentes soluções exploradas, e pela escolha da solução de melhor custo.

Na literatura são encontrados vários métodos que podem ser usados para gerar codificações de prioridade (por exemplo, algoritmos genéticos, "simulated annealing", etc). O algoritmo proposto por Aiken, Nicolau e Novack [27] - devidamente adaptado em [5] para explorar soluções alternativas - é utilizado neste trabalho, para construir soluções a partir de codificações de prioridade.

A abordagem originalmente proposta em [5] é descrita nas Seções de 3.1 a 3.3. A extensão da abordagem para capturar as restrições impostas por um número limitado de barramentos é descrita na Seção 3.4.

3.1 - A Decomposição da Abordagem

A abordagem utilizada para resolver os problemas de escalonamento definidos na Seção 2.2.3 pode ser descrita como a interação entre dois blocos principais: o *explorador* e o *construtor*.

Uma visão geral dessa abordagem é mostrada na Figura 3.1. Pode-se observar nesta figura a independência entre os blocos e a comunicação entre os mesmos através dos parâmetros ? e Custo.

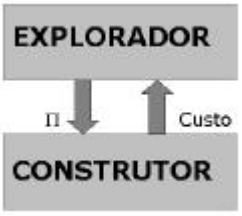


Figura 3.1 – Visão geral da abordagem

O explorador é o bloco encarregado da criação da codificação de prioridade das operações a serem escalonadas. Essa codificação é definida como uma permutação ? das operações do DFG. A ordem das operações em ? define sua prioridade relativa. Assim, no caso de mais de uma operação estar pronta para escalonamento, sua prioridade em ? determina qual delas é selecionada.

O construtor tem a função de gerar uma solução a partir de uma permutação ? obtida do explorador, através do escalonamento das operações do DFG e pelo cálculo do custo da solução. Neste trabalho, o custo da solução é a latência do SMG.

Observe que o construtor limita-se a interpretar codificações de prioridade em ? - mas não as altera, pois somente o explorador pode fazê-lo.

Esta abordagem modular com a separação das funcionalidades em blocos diferentes, tem a vantagem de permitir a adaptação do algoritmo em um dos blocos sem interferir na funcionalidade do outro, por exemplo.

Outra vantagem desta abordagem é a probabilidade de se alcançar um compromisso entre a qualidade de uma solução e o esforço computacional para obtê-la, através de um parâmetro definido pelo usuário que limita o número de codificações de prioridade geradas, facilitando o controle do número de soluções que se deseja explorar.

O bloco explorador não faz parte do escopo deste trabalho. Este é objeto de trabalho correlato, no âmbito do Projeto DESERT/OASIS, visando a incorporação de meta-heurísticas para melhorar o processo de exploração de soluções. Para os experimentos deste trabalho, o explorador limita-se a gerar codificações de prioridade aleatoriamente.

Segue-se detalhando a abordagem, explicando a codificação de prioridades na Seção 3.2 e refinando a descrição do bloco construtor na Seção 3.3.

3.2 - A Codificação de Prioridade

Como mencionado anteriormente, a codificação de prioridade é definida como uma permutação π das operações do DFG e é tarefa exclusiva do explorador. O explorador não considera as restrições de precedência definidas no DFG quando constrói a permutação π (através de um critério de busca local), pois isso é tarefa do construtor. O explorador limita-se a obter uma ordenação para todas as operações do DFG. Isso se deve ao fato de o escalonador utilizar π apenas como critério de desempate na seleção de operações.

Um critério de desempate é necessário quando o conjunto de operações prontas para o escalonamento em um dado estado for maior que o número de UFs disponíveis naquele estado. Assim, a permutação π determina a resposta para a seguinte pergunta: dentre todas as operações prontas, qual deve ser selecionada para o escalonamento?

A operação ocupando a posição de mais alta prioridade em ? é a selecionada. Por exemplo na Figura 3.2, as operações A e B podem ser executadas simultaneamente em diferentes recursos. Todavia, caso haja apenas um somador no datapath, A será selecionada para ser escalonada no primeiro ciclo, pois A precede B na codificação ? .

Uma operação é considerada pronta para ser escalonada em um dado estado quando todos os seus predecessores já tenham sido escalonados em estados anteriores, e há tempo suficientemente longo para acomodar seus atrasos de execução (vide Definição 2.14).

Por exemplo, na hipótese de que haja um único somador, **C** só estará pronta para se escalonada quando as operações **A** e **B** tiverem terminado sua execução (após 2 ciclos de relógio).

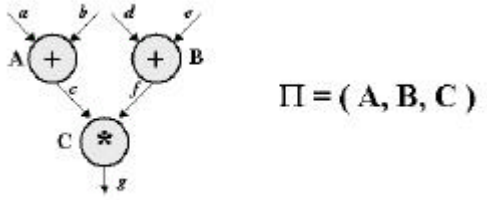


Figura 3.2 – Exemplo de codificação de prioridade

A noção de codificação de prioridade pode ser assim formalizada:

Dado um DFG = (V, E), a codificação de prioridade determinada por essa abordagem é a permutação ? das operações V. A noção de prioridade é associada com a respectiva posição de permutação. Assim ? (v_i) denota a posição de uma operação v_i na permutação ? . Diz-se que v_i precede v_j na permutação ? , matematicamente escrito como $v_i <_? v_j$, se ? (v_i) < ? (v_j).

3.3 - O Construtor de Soluções

O construtor consiste em dois blocos principais: um *escalonador* e um *paralelizador*, como mostra a Figura 3.3. Pode-se observar na figura a independência entre esses dois blocos e os parâmetros de comunicação entre eles, que serão explicados a seguir.

Figura 3.3 – Detalhamento do construtor

3.3.1 - O Paralelizador

A função do paralelizador é capturar o paralelismo entre as operações da descrição comportamental. Num caso mais genérico, esse paralelismo deve ser buscado além das fronteiras impostas por construções condicionais ou laços de iteração, os quais foram objeto de outros trabalhos no âmbito do Projeto DESERT/OASIS [6, 7].

Na sua essência, o paralelizador cria um estado atual (s_k) onde as operações serão escalonadas. Esgotadas as operações passíveis de escalonamento no estado atual, o paralelizador cria um novo estado (s_{k+1}), chamado de próximo estado. Dessa forma o SMG vai sendo criado pelo paralelizador passo a passo, conforme vão se esgotando os recursos para acomodar operações em um dado estado, até que não restem mais operações a escalonar.

O paralelizador também mantém atualizado, a cada transição para um novo estado, o conjunto de operações prontas para serem nele escalonadas, ou seja, as *operações prontas*, conforme a Definição 2.14. O conjunto de operações prontas para escalonamento em um estado s_k é denotado por A_k. Todas as operações em A_k poderiam ser executadas em paralelo, caso houvesse recursos suficientes para acomodá-las. A indisponibilidade de recursos ilimitados requer a seleção de operações de maior prioridade. O conjunto A_k é implementado de forma a manter seus elementos ordenados de acordo com a codificação de prioridade ? [6].

O bloco paralelizador controla a criação de estados no SMG, compondo a máquina de estados finitos que descreve em qual estado cada operação é executada. À medida que as operações são retiradas do conjunto A_k (operações prontas para escalonamento) e colocadas na lista OP_k (operações efetivamente escalonadas), o paralelizador anota no SMG que tais operações foram escalonadas no estado s_k e, caso seja necessário, cria um próximo estado.

Encontrar operações prontas para serem escalonadas é a principal tarefa do paralelizador. Na ausência de construções condicionais e laços, a determinação das operações prontas é realizada observando-se os sucessores das operações escalonadas no estado atual s_k, denotados por $SUCC_k$. Dado um elemento de $SUCC_k$, caso todos os seus predecessores no DFG já tenham sido escalonados, então este elemento é uma operação pronta para ser escalonada no próximo estado.

3.3.2 - O Escalonador

A função do escalonador é selecionar - do conjunto A_k - as operações v_i para serem executadas no estado s_k. Essa seleção é realizada obedecendo a codificação de prioridade ? e a quantidade de recursos disponíveis. O escalonador então retorna um conjunto de operações escalonadas no estado atual, representada pelo conjunto OP_k, para que o paralelizador as retire do conjunto s_k, nele adicionando as operações que se tornaram prontas. Isso é repetido até que todos os recursos disponíveis são ocupados no estado s_k, ou

até que o conjunto A_k esteja vazio, indicando que todos os vértices do DFG foram escalonados.

O bloco escalonador atua selecionando as operações que serão escalonadas e gerenciando a disponibilidade dos recursos físicos. Como as operações prontas estão permanentemente ordenadas de acordo com a codificação de prioridade ? , o escalonador seleciona sempre a operação de mais alta prioridade que satisfaça a restrição de recursos.

A seleção das operações é dependente apenas da prioridade definida no explorador e da disponibilidade de recursos. Não depende da heurística do escalonador, como na maior parte dos métodos clássicos.

3.4 - A Extensão da Abordagem

Esta seção descreve os algoritmos que implementam o bloco construtor. Em especial mostra-se como a modelagem proposta neste trabalho – para as restrições impostas pelos barramentos – é integrada à abordagem utilizada em trabalhos anteriores [4, 6 e 7].

Os quatro primeiros algoritmos a serem descritos capturam a modelagem proposta para as restrições impostas por barramentos (veja Seção 2.3). Os demais algoritmos mostram como nossa modelagem se integra à abordagem, estendendo o escalonador para suportar restrições impostas por barramentos.

O Algoritmo 3.1 percorre os componentes do vetor de barramentos **b** e retorna o primeiro componente livre, se houver, conforme a Definição 2.16.

```
EncontreBarramentoLivre (b)
  {
      para cada b_i ? b faça
          {
           se (b_i = nenhum) então
          retorne (i);          // o i-ésimo barramento está livre
        }
           retorne (0);          // não há barramentos livres
  }
```

Algoritmo 3.1 – Busca de um barramento livre

Dado um operando arbitrário a ser consumido por uma dada operação, o Algoritmo 3.2 seleciona um barramento que já esteja sendo utilizado por alguma RT potencialmente compatível ou, caso não exista, seleciona um barramento livre.

```
SelecioneBarramento (operando, b)
  {
      se (? b_i? b | operando = b_i)
          então retorne (i);
      senão
          retorne (EncontreBarramentoLivre (b));
  }
```

Algoritmo 3.2 – Seleção de um barramento

O Algoritmo 3.3 constrói uma RT compatível com outras RTs já construídas. Essencialmente, dada uma operação v, o algoritmo anota nos componentes selecionados do vetor de ocupação de barramentos os operandos e o resultado da operação v. Isso resulta em um mapeamento de valores para barramentos, conforme a Definição 2.17.

Como a seleção de barramentos obedece à Definição 2.18, a RT gerada pelo Algoritmo 3.3 é compatível por construção com quaisquer outras RTs que implementem operações executadas no mesmo estado.

```
ConstruaRT (v, b)
{
        i ? SelecioneBarramento (v.operando1);
        j ? SelecioneBarramento (v.operando2);
        k ? SelecioneBarramento (v.resultado);
        se (i ? 0 ? j ? 0 ? k ? 0) então
        {
           bi = v.operando1;
           bj = v.operando2;
           bk = v. resultado;
           retorne (verdadeiro);
        }
         senão
           retorne (falso);
}
```

Algoritmo 3.3 – Construção de uma RT compatível

O Algoritmo 3.4 reinicializa a ocupação dos barramentos. Ele será utilizado antes de se escalonar um novo estado do SMG.

```
LibereBarramentos (b)
{
        para cada bi ? b faça
           bi = nenhum;
}
```

Algoritmo 3.4 – Liberação de barramentos

O Algoritmo 3.5 restringe o conjunto de operações prontas A_k retornando o subconjunto P de operações prontas cujo escalonamento resultaria em RTs incompatíveis e removendo-as do conjunto A_k.

```
RestrinjaOperaçõesProntas (Ak, b)
{   P = ?
    para cada vi ? Ak faça
    {
       se ConstruaRT (vi, b ) = falso então
             retire vi de Ak;
       insira vi em P;
    }
```

```
    retorne P;
}
```

Algoritmo 3.5 – Restrição das operações prontas devido ao número limitado de barramentos

O Algoritmo 3.6 seleciona a operação pronta v_i de maior prioridade que obedece à restrição imposta pelo número limitado de unidades funcionais (**a**).

```
SelecioneOperação (Ak, a, ? )
{
        escolha a operação vi ? Ak com maior prioridade ? que satisfaz a restrição a;
        retorne (vi);
}
```

Algoritmo 3.6 – Seleção da operação a ser escalonada

Dado um estado atual s_k, o Algoritmo 3.7 nele escalona operações enquanto houver operações prontas ou recursos disponíveis, e ao final, retorna o conjunto OP_k de operações naquele estado.

```
EscaloneEstado (sk, Ak, a, ? )
{
        OPk = ? ;
        vi = SelecioneOperação (Ak, a, ? );
        enquanto (vi ? nenhuma) faça {
                escalone vi em sk ;
                OPk = OPk ? {vi};
                Retire vi de Ak ;
        }
        retorne ( OPk );
}
```

Algoritmo 3.7 – Função que escalona estados em A_k

Dadas as operações escalonadas em um estado s_k, o Algoritmo 3.8 retorna as operações que se tornam prontas (P), como conseqüência das operações escalonadas naquele estado (OP_k). A idéia básica é que uma operação torna-se pronta quando todos os seus predecessores já foram escalonados.

```
EncontreOperaçõesProntas (OPk)
{
        para cada u ? OPk
                para cada sucessor v de u no DGF
                        se (todos os predecessores de v no DFG foram escalonados)
                                P = P ? {v};
        retorne (P);
}
```

Algoritmo 3.8 – Cálculo das operações prontas

O Algoritmo 3.9 mostra como o Construtor gera uma solução para uma dada codificação de prioridade ? obedecendo às restrições impostas pelo número limitado de unidades funcionais (**a**) e barramentos (**b**).

```
Construtor ( a, b, ? )
{
        ? = 0 ;
        crie estado inicial so no SMG;
        A0 = { vi ?v0 é o único predecessor de vi no DFG } ;
        próximo = so;
        enquanto ( próximo ? nenhum ) faça
          {  sk = próximo ;
             P = RestrinjaOperaçõesProntas (Ak, b);
             OPk = EscaloneEstado  (sk, Ak, a, ? ) ;
             ? = ? + 1;
             P = P ? EncontreOperaçõesProntas (OPk) ;
             P = P ? Ak ;
             Se ( P ? ? )
                {
                   crie um novo estado sk+1 no SMG;
                   Ak+1 = P ;
                   próximo = sk+1 ;
                   P = ? ;
                   LibereBarramentos (b);
                }
                senão
```

Algoritmo 3.9 – A construção de uma solução

A inicialização do procedimento **Construtor** (**a**, **b**, ?) consiste em zerar a latência, criar um estado inicial e calcular as operações prontas para serem nele escalonadas, ou seja, aquelas que têm como único predecessor o pólo fonte v_0. Esse procedimento escalona um estado atual s_k associando tantas operações quantas puderem ser acomodadas nos recursos durante um ciclo de relógio. Esgotadas as operações passíveis de escalonamento em s_k, esse procedimento cria o próximo estado s_{k+1}, que será o estado atual na próxima iteração do laço. Assim, operações são escalonadas em estados sucessivos, até que não restem mais operações a serem escalonadas.

Como um estado corresponde a um ciclo de relógio, a latência ? é incrementada de um a cada novo estado escalonado.

Essencialmente, os algoritmos acima descritos foram implementados visando agregar as restrições impostas pelos barramentos, ao construtor apresentado em [6].

Para a implementação utilizou-se a plataforma de trabalho do Projeto DESERT/OASIS: microcomputador PC, sistema operacional Linux, ambiente de trabalho KDE, ambiente de desenvolvimento Kdevelop e linguagem de programação C++.

No próximo capítulo apresentam-se os experimentos realizados com o protótipo implementado.

4 Resultados Experimentais

Este capítulo apresenta os resultados obtidos nos experimentos com a implementação dos algoritmos vistos na Seção 3.4.

O objetivo dos experimentos relatados a seguir é mostrar o impacto quantitativo na latência devido a um número pré-fixado de barramentos.

Para validação e avaliação da técnica proposta, utiliza-se três exemplos clássicos da literatura em HLS: *DIFFEQ* [1], o qual é o algoritmo de um integrador para equações diferenciais; *WDELF* [27], o qual é o algoritmo que implementa um filtro de onda digital de quinta ordem; e *FDCT* [28], o qual é o algoritmo que propicia compactação de imagens através de transformação discreta de co-senos. A Tabela 4.1 resume as principais características dos DFGs resultantes dos exemplos adotados neste trabalho para fins de "benchmarking", cujas representações tabulares e gráficas estão anexas a este trabalho.

Exemplo	Vértices	Arestas	Tipo de operações			Operandos
			+	-	*	
DIFFEQ	11	16	3	2	6	10
WDELF	36	66	25	1	8	49
FDCT	44	68	13	13	16	61

Tabela 4.1 – Resumo das características dos exemplos usados como benchmarks

Nas colunas de vértices e de arestas da Tabela 4.1 estão computados os pólos dos grafos (fonte e sumidouro), bem como todas as arestas emergentes do vértice fonte e as incidentes no vértice sumidouro. Esses vértices e arestas foram associados a atrasos nulos de execução.

A última coluna exibe o número de operandos distintos utilizados em cada exemplo. Assume-se que as operações de adição, comparação e subtração executam em UFs do tipo unidade lógico-aritmética (ALU). As operações de multiplicação executam em UFs do tipo multiplicador (MUL).

Para cada exemplo, foram construídas 500 soluções, induzidas através de codificações de prioridade (?) geradas aleatoriamente. Em cada exemplo, mediu-se a latência (?) de cada solução para diferentes restrições (números de UFs e barramentos).

Nas tabelas a seguir, para cada exemplo e cada restrição, reporta-se a menor latência encontrada para o conjunto de soluções geradas.

Sem perda de generalidade, os experimentos aqui descritos foram realizados sob a hipótese de que o atraso de execução de todas as operações é unitário.

A Tabela 4.2 compara os resultados com os obtidos pelo método descrito em [6]. A primeira coluna rotula diferentes casos associados às distintas restrições de UFs apresentadas na segunda coluna. A última coluna ilustra as menores latências obtidas sob a hipótese de um número ilimitado de barramentos. As colunas intermediárias mostram as menores latências observadas sob diferentes restrições de barramentos.

CASO	Nº de UFs	Nº de Barramentos									
		3	4	5	6	7	8	9	10	15	ilimitado
A	1 MUL 1 ALU	34	34	33	27	27	27	27	27	27	27
B	1 MUL 2 ALU	34	34	30	19	19	18	16	16	16	16
C	2 MUL 2 ALU	34	34	29	19	19	17	16	16	16	16
D	3 MUL 3 ALU	34	34	29	19	18	17	15	15	14	14

Tabela 4.2 – Latências para o exemplo WDELF

Para melhor visualização, os valores da Tabela 4.2 são mostrados no Gráfico 4.1 na forma de diagrama de barras verticais.

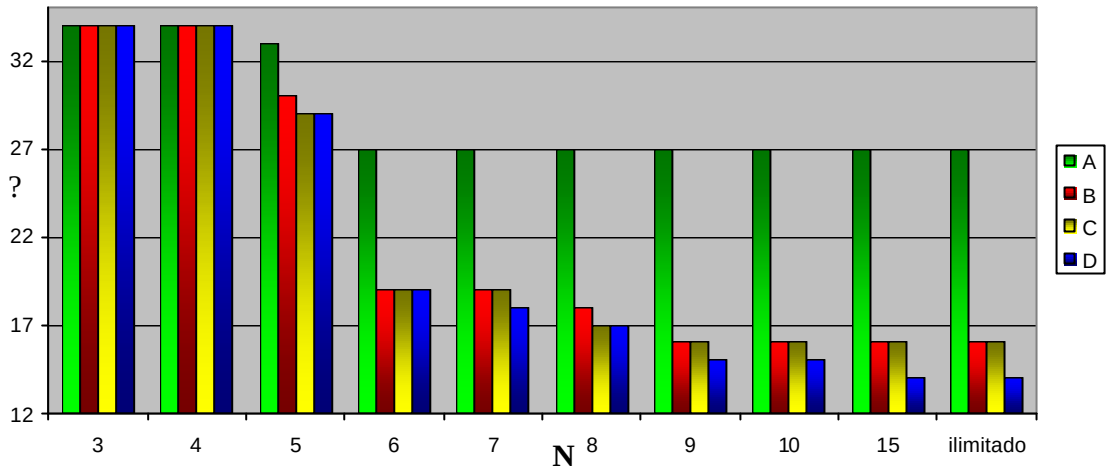


Gráfico 4.1 – Relação latência versus restrição de barramentos para WDELF

Como esperado, para um mesmo número de UFs, a latência diminui com o aumento do número de barramentos até atingir a latência obtida sob a hipótese de um número ilimitado de barramentos. A partir de um determinado ponto, por mais que se aumente o número de barramentos, a latência não diminui, pois já há barramentos suficientes para transportar todos os dados.

Nota-se que, para um número pequeno de barramentos, a latência obtida independe do número de UFs, pois embora o paralelismo possa ser nelas acomodado, ele não é viabilizado pela escassez de barramentos para transportar operandos para as UFs. Pela mesma razão, a variação da latência relacionada com o número de barramentos aumenta, para um número maior de UFs.

Ainda na Tabela 4.2, observa-se que há um limite superior para o número de barramentos necessários para atingir a menor latência. Assumindo que todos os operandos consumidos sejam distintos, esse limite é igual a 3 vezes o número de UFs (dois operandos e um resultado). Por exemplo, no caso A o número de barramentos necessários para atingir ? = 27 é igual a 2 x 3 = 6.

Entretanto, o número de barramentos para atingir a menor latência pode ser ligeiramente menor que o limite superior, devido à compatibilidade de RTs (casos C e D). Por exemplo, no caso D, 3x(3+3) = 18 barramentos seriam necessários no pior caso, mas 15 barramentos são suficientes para atingir ? = 14, ou seja, 83% do limite superior. Por outro lado, um número excessivamente pequeno de barramentos acaba inviabilizando a compatibilidade de RTs e o número de barramentos corresponde ao limite superior (casos A e B).

CASO	Nº de UFs	Nº de Barramentos						
		3	4	5	6	10	15	ilimitado
A	1 MUL 1 ALU	11	11	8	7	7	7	7
B	2 MUL 1 ALU	11	10	8	6	5	5	4
C	2 MUL 2 ALU	11	10	8	5	4	4	4
D	3 MUL 1 ALU	11	10	8	6	5	5	4
E	3 MUL 2 ALU	11	10	8	5	4	4	4
F	4 MUL 1 ALU	11	10	8	6	5	5	4

Tabela 4.3 – Latências para o exemplo DIFFEQ

A Tabela 4.3 tem estrutura similar à Tabela 4.2, mas refere-se ao exemplo DIFFEQ. De forma também análoga, o Gráfico 4.2 mostra os resultados da Tabela 4.2 em forma de diagrama de barras.

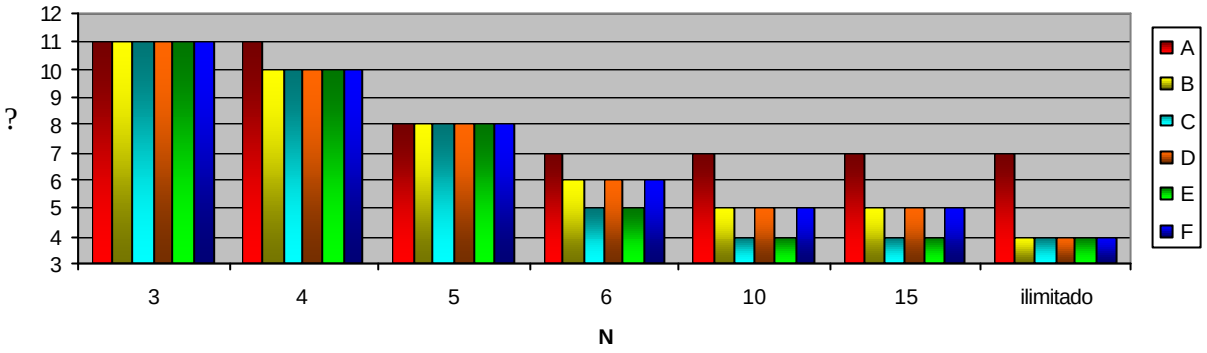


Gráfico 4.2 – Relação latência versus restrição de barramentos para DIFFEQ

A Tabela 4.4 também tem estrutura similar à Tabela 4.2, mas refere-se ao exemplo FDCT. De forma também análoga, o Gráfico 4.3 mostra os resultados da Tabela 4.3 em forma de diagrama de barras.

CASO	Nº UFs	Nº de Barramentos									
		3	4	5	6	8	9	12	16	18	ilimitado
A	1 MUL 1 ALU	41	41	37	31	31	31	31	31	31	31
B	1 MUL 2 ALU	41	34	25	21	16	16	16	16	16	16
C	2 MUL 3 ALU	41	34	25	21	14	13	11	11	11	11
D	3 MUL 3 ALU	41	34	25	21	14	13	11	11	11	11
E	3 MUL 5 ALU	41	34	25	21	14	13	10	8	8	7
F	4 MUL 8 ALU	41	34	25	21	14	13	9	7	7	6
G	8 MUL 4 ALU	41	34	25	21	14	12	10	8	6	6

Tabela 4.4 – Latências para o exemplo FDCT

Observe que os resultados das Tabelas 4.3 e 4.4 (e os respectivos Gráficos 4.2 e 4.3) apresentam comportamento qualitativamente similar ao do primeiro exemplo, no que se refere à variação de latência.

Entretanto, o exemplo FDCT exibe um impacto quantitativo mais pronunciado que os demais. Por exemplo, no caso G da Tabela 4.4, embora o limite superior para o número de barramentos seja 3x(8+4) = 36, o menor número de barramentos para atingir a latência mínima é 18, ou seja, 50% do limite superior. Isso pode ser assim interpretado: em média a metade dos valores de operandos e resultados são compartilhados pelas operações.

?

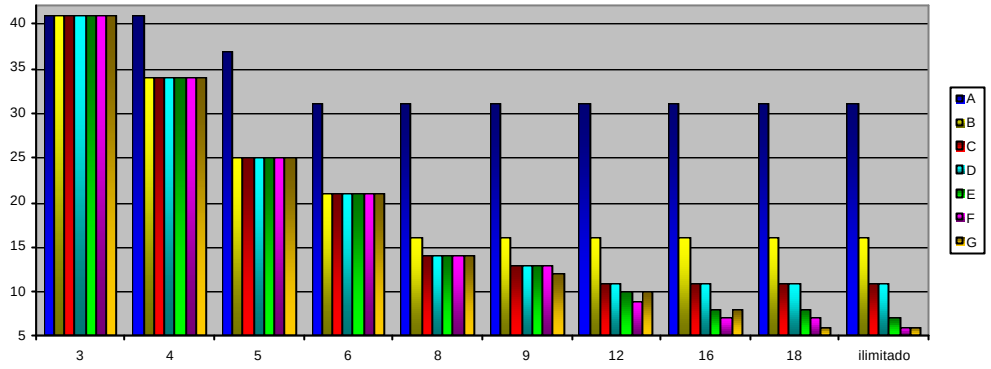


Gráfico 4.3 – Relação latência versus restrição de barramentos para FDCT

N

O próximo capítulo apresenta as conclusões e sugestões para a continuidade do trabalho de pesquisa.

5 Considerações Finais

5.1 - Conclusões

Este trabalho mostrou como se pode modelar restrições impostas por um número limitado de barramentos e como integrar essa modelagem no âmbito de uma abordagem que permite a construção e a exploração de soluções alternativas para o problema de escalonamento sob restrição de recursos.

A modelagem proposta estende a abordagem original [5], a qual se limita ao tratamento de restrições impostas por unidades funcionais (Problema 2.1), permitindo também a avaliação do impacto de um número limitado de barramentos (Problema 2.2).

Os resultados experimentais indicam que o modelo é consistente. Ademais, para um número pré-fixado de unidades funcionais M, os resultados mostram que o número de barramentos suficiente para preservar a latência mínima pode ser inferior a 3M, se operandos comuns puderem ser escalonados simultaneamente. Ora, essa habilidade de otimização é produto da formalização da relação de compatibilidade de RTs e de sua verificação durante o escalonamento. Em outras palavras, tal modelagem orienta o escalonador para não superestimar o impacto dos barramentos na latência.

5.2 - Trabalhos Futuros

5.2.1 - Extensão da habilidade de exploração

O estágio atual deste trabalho permite a exploração de alternativas arquiteturais no que se refere ao número de unidades funcionais e o número de barramentos. Seria desejável que a abordagem fosse também capaz de avaliar o impacto de um número pré-fixado de registradores. Por essa razão, a continuidade da pesquisa descrita nesta dissertação de mestrado deveria passar pela solução de um problema mais geral, a seguir enunciado:

Problema 5.1 - Dado um grafo de fluxo de dados DFG (V, E), um vetor de restrição **a**, um número de barramentos **N** e um número de registradores **R**, encontre um escalonamento ? que minimize a latência ?.

A solução desse problema mais geral permite a exploração do espaço de soluções em função dos três parâmetros que definem a estrutura do "datapath".

5.2.2 - Extensão para modelagem de restrições de tempo

Há um trabalho correlato [9] que aborda a modelagem de restrições de tempo. Esse trabalho pressupõe apenas restrições impostas por UFs. A combinação das idéias daquele trabalho com a modelagem aqui proposta resultaria em um sistema capaz de tratar restrições impostas por requisitos de tempo real, restrições de unidades funcionais, de barramentos e de registradores. Por razões de limitação de tempo, tal extensão não estava no escopo desta dissertação, mas fica aqui sugerida como trabalho futuro.

6 Referências Bibliográficas

[1] DE MICHELI, Giovanni, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, pp. 1-5, 12-27, 1994.

[2] MURDOCCA, Miles J., *Introdução à Arquitetura de Computadores*, Ed. Campus, pp. 202-203,224-262, 2000.

[3] WOLF, W., *Computers as Components*, MK Publishers, pp. 57-60, 10-21, 2000.

[4] CAMPOSANO,R., *"Path-based scheduling for synthesis"*, IEEE Trans. On Computer-Aided Design, Vol 10 nº 1, 1991.

[5] SANTOS, Luiz C. V., *"Exploiting instruction-level parallelism: a constructive approach"*, Eindhoven University of Technology, Eindhoven – Holanda, PhD. Thesis, 1998.

[6] AZAMBUJA, Rogério X., "*Escalonamento e Otimização de Registradores para Grafos de Fluxo de Dados*" –Trabalho Individual. Universidade Federal de Santa Catarina, Florianópolis – Brasil, 2001.

[7] FERRARI, Dione J.; *"Aplicação de Loop Pipelining e Loop Unrolling à Síntese de Alto Nível"* – Dissertação de Mestrado. Universidade Federal de Santa Catarina, Florianópolis – Brasil, 2002.

[8] TIMMER, Adwin H., *"From Design Space Exploration to Code Generation:a Constraint Satisfaction Approach for the Architectural Synthesis of digital VLSI Circuits"* – Tese de Doutorado. Eindhoven University, Eindhoven – Holanda, 1996.

[9] TOLENTINO, C. H., "*Modelagem e Análise de Restrições de Tempo Real no Escalonamento em HLS"* – Trabalho Individual. Universidade Federal de Santa Catarina, Florianópolis – Brasil, 2003.

[10] BRINGMANN, O., e ROSENSTIEL, W.,*"Cross-Level Hierarchical High-Level Synthesis"* , DATE, Paris – França, 1998.

[11] LI, Jan e GUPTA, R. K, *"An Algorithm to Determine Mutially Exclusive Operations in Behavioral Descriptions"*, DATE, Paris – França, 1998.

[12] SAFARI, S., ESMAEILZADEH, H. e JAHANGIR A., *"A Novel Improvement Technique for High-Level Synthesis"*, ISCAS, Bagkok – Tailandia, 2003.

[13] ERCANLI, E., e PAPACHRISTOU, C., *"A Register File and Scheduling Model for Application Specific Processor Synthesis"*, 33rd DAC, Las Vegas – USA, 1996.

[14] FRANK, E., RAJE, S. e SARRAFZADEH, M., *"Constrained Register Allocation in Bus Architectures"*, 32nd DAC, San Francisco – USA, 1995.

[15] SO, B., DINIZ, P.C., e HALL, M.H., *"Using Estimates from Behavioral Synthesis Tools in Compiler-directed Design Space Exploration"*, 40th DAC, Anaheim – USA, 2003.

[16] RUSSEL, J., e JACOME, F. M., *"Architecture-Level Performance Evaluation of Component-based Embedded Systems"*, 40th DAC, Anaheim – USA, 2003.

[17] TANEMBAUM, A. S., *Redes de Computadores*, Editora Campus, Rio de Janeiro – Brasil, pp. 57-58, 1999.

[18] ELES, P., KUCHCINSKI, K., PENG, Z., DOBOLI, A., e POP, P., *"Scheduling of Conditional Process Graphs for the Synthesis of Embedded Systems"*, DATE, Paris – França, 1998.

[19] RUST, C., STAPPERT, F., ALTENBERND, P., e TACKEN, J., *"From High-level Specifications down to Software Implementations of Parallel Embedded Real-time Systems"*, DATE, Paris – França, 2000.

[20] MEYEROWITZ, T., PINELLO, C., e SANGIOVANNI-VICENTELLI, A., *"A Tool for Describing and Evaluating Hierarchical Real-Time Bus Scheduling Policies"*, 40th DAC, Anaheim – USA, 2003.

[21] HAYNAL, S., e BREWER F., *"A Model for Scheduling Protocol-Constrained Components and Environments"*, 36th DAC, New Orleans – USA, 1999.

[22] KUNG, S. Y., WHITEHOUSE, H. J., e KAILATH, T., *VLSI and Modern Signal Processing*, Prentice Hall, 1985.

.

[23] HEIJLINGERS, M. J. M., *"The Application of Genetic Algorithms to High-Level Synthesis"*, PhD Thesis, Eindhoven University, Eindhoven – Holanda, 1996. p. 116.

[24] AMELLAL, S., e KAMINSKA, B., *"Functional Synthesis of Digital Systems with TASS"*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 13, Nº5, 1994.

[25] LEIJTEN, J. A. J., VAN MEERBERGEN, J. L., TIMMER, e A. H., JESS, A. G., *"Stream Communication between Real-Time Tasks in a High Performance Multiprocessor"*, DATE, Paris – França, 1998.

[26] DI NATALE, M., SANGIOVANNI-VICENTELLI, e A., BALARIN, F.,*"Task Scheduling with RT Constraints"*, 37th DAC, Los Angeles - USAl, 2000.

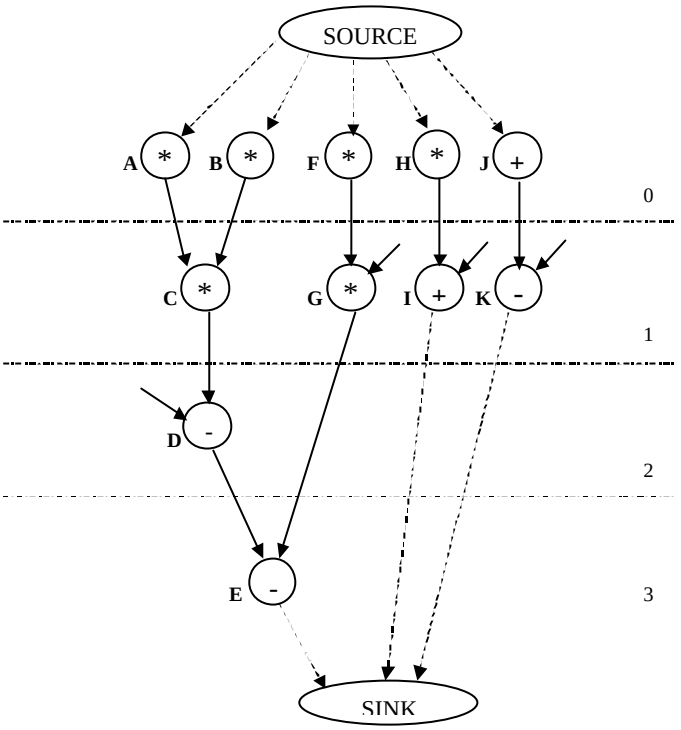
[27] AIKEN, A., NICOLAU, A., e NOVACK, *"Resource-Constraint Software Pipelining"*, IEEE Transactions on Paralel and Distributed Systems, vol. 6, Nº 12, pp. 1248-1270, December, 1995.

[28] MALLON, D. J.; DENYER, P. B., A New Approach to PipelineOptimization, EDAC'90, Glasgow – Escócia, 1990.

A N E X O S

Anexo I – Descrição tabular e representação gráfica do DFG para o exemplo DIFFEQ

Operação	Arestas incidentes		Resultado
	Operando 1	Operando 2	
A	a	b	i
B	c	d	j
F	a	e	k
H	c	d	l
J	b	d	f
C	i	j	m
C	i	j	m
G	k	d	n
I	e	l	p
K	f	g	q
D	c	m	o
E	o	n	h
E	o	n	h

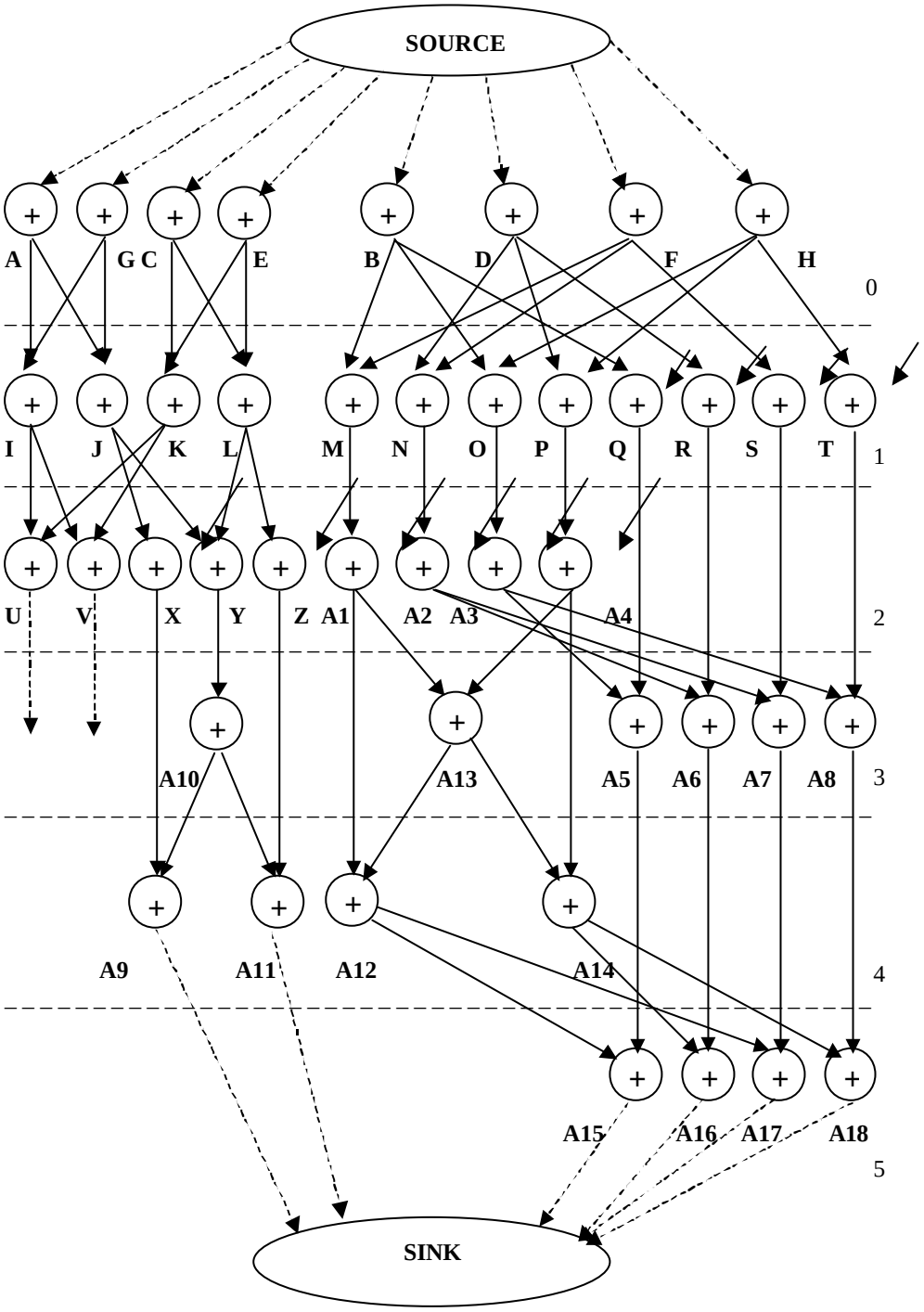

SOURCE
A
*
B
*
F
*
H
*
J
+
0
C
*
G
*
I
+
K
-
1
D
-
2
E
-
3
SINK

Anexo II – Descrição tabular do DFG para o exemplo FDCT

Operação	Arestas incidentes		Resultado
	Operando 1	Operando 2	
A	x0	x7	x8
B	x0	x7	x9
C	x1	x6	x10
D	x1	x6	x11
E	x2	x5	x12
F	x2	x5	x13
G	x3	x4	x14
H	x3	x4	x15
I	x8	x14	x16
J	x8	x14	x17
M	x9	x13	x20
O	X9	x15	x22
Q	x9	c1	x24
K	x10	x12	x18
L	x10	x12	x19
N	x11	x13	x21
P	x11	x15	x23
R	x11	c2	x25
K	x10	x12	x18
L	x10	x12	x19
M	x9	x13	x20
N	x11	x13	x21
S	c3	x13	x26
I	x8	x14	x16
J	x8	x14	x17
O	x9	x15	x22
P	x11	x15	x23
T	c4	x15	x27
U	x16	x18	y0
V	x16	x18	y4
X	x17	c5	x28
Y	x17	x19	x29
U	x16	x18	y0
V	x16	x18	y4

Operação	Arestas incidentes		Resultado
	Operando 1	Operando 2	
Y	x17	x19	x29
Z	c6	x19	x30
A1	c7	x20	x31
A2	c8	x21	x32
A2	c8	x21	x32
A3	c9	x22	x33
A4	c10	x23	x34
A5	x24	x33	x37
A6	x25	x32	x38
A7	x26	x32	x39
A8	x27	x33	x40
A9	x28	x35	y2
A10	x29	c11	x35
A11	x35	x30	y6
A12	x31	x36	x41
A13	x31	x34	x36
A6	x25	x32	x38
A7	x26	x32	x39
A5	x24	x33	x37
A8	x27	x33	x40
A13	x31	x34	x36
A14	x36	x34	x42
15	x37	x41	y1
A16	x34	x42	y3
A17	x39	x41	y5
A18	x40	x42	y7
A9	x28	x35	y2
A11	x30	x35	y6
A15	x37	x41	y1
A17	x39	x41	y5
A12	x31	x36	x41
A14	x34	x36	x42
A16	x38	x42	y3
A18	x40	x42	y7

Anexo III – Representação Gráfica do DFG para o exemplo FDCT



Anexo IV – Descrição tabular do DFG para o exemplo WDELF

Operação	Arestas incidentes		Resultado
	Operando 1	Operando 2	
V	temp6	x28	x35
A4	temp6	x36	x37
A	temp5	temp7	x32
P	temp7	x30	x31
S	temp7	x31	x41
D	temp4	x12	x20
Y	temp3	x10	x15
A5	temp3	x16	x17
B	temp1	p1	x3
C	temp3	x3	x12
A3	p1	x6	x4
E	x32	x20	x25
H	x32	x24	x27
J	x32	x27	x29
C	temp2	x3	x12
Q	x9	x3	x8
U	x8	x3	x7
D	temp4	x12	x20
I	x21	x12	x19
L	x19	x12	x11
E	x20	x32	x25
F	c4	x25	x24
K	x19	x25	x22
G	c1	x25	x21
H	x24	x32	x27
J	x27	x32	x29
R	x27	x31	x28
N	x27	x22	x23
I	x12	x21	x19

Operação	Arestas incidentes		Resultado
	Operando 1	Operando 2	
K	x19	x25	x22
L	x19	x12	x11
T	x19	x8	x10
N	x22	x27	x23
O	c2	x11	x9
Q	x3	x9	x8
M	c5	x29	x30
P	temp7	x30	x31
S	temp7	x31	x41
R	x31	x27	x28
A1	x31	x40	x42
V	temp6	x28	x35
W	c7	x35	x36
A7	x37	x35	x34
A4	temp6	x36	x37
A7	x37	x35	x34
X	c8	x41	x40
A1	x31	x40	x42
T	x8	x19	x10
U	x8	x3	x7
A6	x8	x4	x5
Y	temp3	x10	x15
A8	x17	x15	x14
A2	c6	x15	x16
A5	temp3	x16	x17
A8	x17	x15	x14
Z	c3	x7	x6
A3	p1	x6	x4
A6	x8	x4	x5

Anexo V – Representação Gráfica do DFG para o exemplo WDELF